

Interface Gráfica em Java

Exercícios

Vinicius Takeo Friedrich Kuwaki

Universidade do Estado de Santa Catarina

Seções

Exemplo

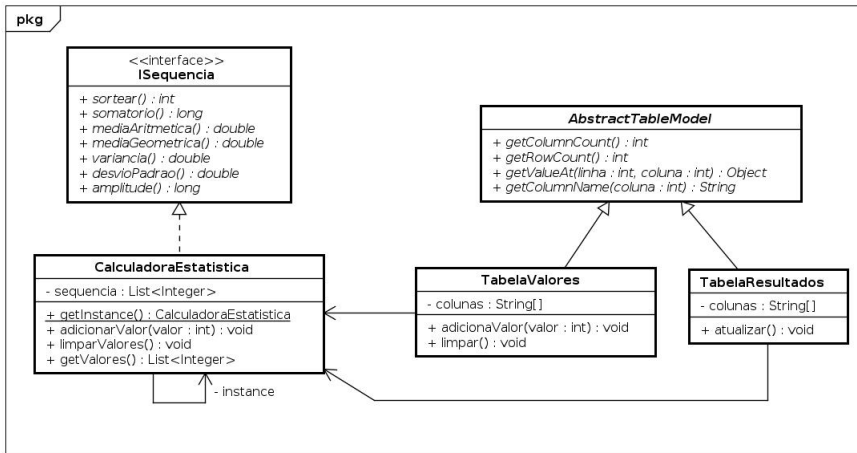
Resolução

Exercício

Exemplo

- A partir da interface *ISequencia* definida na Aula Prática 8: Interfaces em Java, crie uma interface gráfica baseada no diagrama de classes apresentado a seguir:

Exemplo - Diagrama



Exemplo - Descrição

- A classe CalculadoraEstatistica implementa a interface ISequencia;
- Os métodos definidos da interface ISequencia são implementados a partir da manipulação do atributo *valores*.
- O método **adicionarValor()** adiciona um inteiro a lista de valores a ser operada pela Calculadora, enquanto o método **limpar()** remove todos os valores dessa lista.

Exemplo - Descrição

- A partir do exposto, crie uma interface gráfica em Java que faça a entrada de valores para a calculadora e exiba ao usuário os resultados obtidos das chamadas de todos os métodos dessa classe;
- A cada inserção de um novo valor, os resultados devem ser atualizados;
- Caso o usuário deseje, ele poderá limpar a tela que, conseqüentemente, limpa a lista de valores do objeto CalculadoraEstatistica.

Exemplo - Interface Gráfica

- Aqui está o exemplo que será seguido;

The screenshot shows a software window titled "Calculadora Estatística". At the top, there is a table with statistical data. Below this, on the left, is an input area with the label "Digite um valor:" followed by a text box and two buttons labeled "adicionar" and "limpar". On the right, there is a list box titled "Valores" containing the numbers 20 and 40.

Sorteado	Somatorio	Média Aritmética	Média Geométrica	Variancia	Desvio Padrão	Amplitude
40	60	30.0	28.28427124746...	200.0	14.14213562373...	20

Digite um valor:

adicionar

limpar

Valores

- 20
- 40

Seções

Exemplo

Resolução

Exercício

Classe CalculadoraEstatistica

- Vamos começar pelo pacote de dados;
- O código-fonte da interface ISequencia está disponível
<https://github.com/takeofriedrich/monitoriapoo/blob/master/Aulas%20Práticas/Aulas%20Interface%20em%20Java/ISequencia.java>.
- Faça download dele e cole dentro do seu projeto;
- Primeiro vamos implementar a interface ISequencia na classe CalculadoraEstatistica;
- Ela utilizará as bibliotecas Random, List e LinkedList do Java:

```
package dados;  
  
import java.util.LinkedList;  
import java.util.List;  
import java.util.Random;
```

Classe CalculadoraEstatistica

- A classe irá implementar a interface ISequencia, então colocaremos a palavra reservada **implements** após a declaração da classe;
- Como ela é um Singleton, teremos o atributo estático do mesmo tipo da própria classe chamado **instance**, um construtor privado e um método estático **getInstance()**:

```
public class CalculadoraEstatistica implements ISequencia {  
    private static CalculadoraEstatistica instance = null;  
    private CalculadoraEstatistica() {  
    }  
    public static CalculadoraEstatistica getInstance() {  
        if (instance == null) {  
            instance = new CalculadoraEstatistica();  
        }  
        return instance;  
    }  
}
```

Classe CalculadoraEstatistica

- Teremos um atributo do tipo List de inteiros chamado *sequencia*;
- Para esse atributo teremos os métodos get e set, além de um método que adiciona um inteiro nessa lista e um que remove todos os valores dela:

```
private List<Integer> sequencia = new LinkedList<Integer>();

public void adicionarValor(int valor) {
    this.sequencia.add(valor);
}

public List<Integer> getValores() {
    return sequencia;
}

public void setValores(List<Integer> valores) {
    this.sequencia = valores;
}

public void limparValores() {
    this.sequencia.clear();
}
```

Classe CalculadoraEstatistica

- Os métodos da interface ISequência já foram implementados na Aula Prática 8: Interfaces em Java;
- Vamos apenas copiar e colar a implementação desses métodos da Aula 8:

```
public int sortear() {  
    final Random r = new Random();  
    return sequencia.get(r.nextInt(sequencia.size()));  
}
```

```
public long somatorio() {  
    long soma = 0;  
    for (final int x : sequencia) {  
        soma += x;  
    }  
    return soma;  
}
```

Classe CalculadoraEstatistica

```
public long produtorio() {  
    long produto = 1;  
    for (final int x : sequencia) {  
        produto = produto * x;  
    }  
    return produto;  
}
```

```
public double mediaGeometrica() {  
    return Math.pow(produtorio(), 1.0 / (double) (sequencia.size()));  
}
```

```
public double mediaAritmetica() {  
    return somatorio() / (double) (sequencia.size());  
}
```

```
public double variancia() {  
  
    final double media = mediaAritmetica();  
    double soma = 0;  
  
    for (final int x : sequencia) {  
        soma += Math.pow(x - media, 2);  
    }  
  
    return soma / ((double) (sequencia.size() - 1));  
}
```

Classe CalculadoraEstatistica

```
public double desvioPadrao() {  
    return Math.sqrt(variancia());  
}
```

```
public int amplitude() {  
  
    int maior = sequencia.get(0);  
    int menor = sequencia.get(0);  
  
    for (final int x : sequencia) {  
        if (x > maior) {  
            maior = x;  
        }  
        if (x < menor) {  
            menor = x;  
        }  
    }  
  
    return maior - menor;  
}
```

Classe TabelaResultados

- Terminado o pacote de dados vamos para o de apresentação;
- Começaremos pela tabela de resultados;
- Ela estende a classe abstrata *AbstractTableModel*, da biblioteca Swing;
- A qual requer a implementação de 3 métodos abstratos:
 - **int getColumnCount():** retorna o número de colunas da tabela;
 - **int getRowCount():** retorna o número de linhas da tabela;
 - **Object getValueAt(int linha, int coluna):** retorna o objeto a ser exibido em uma célula da tabela;
- Vamos também sobreescrever o método **String getColumnName(int coluna)** para colocar o nome das colunas da tabela;
- Por *default* a classe *AbstractTableModel* retorna como nome das colunas A,B,C,...;
- Atente-se ao fato de que a JTable apenas irá exibir o nome das colunas utilizando o método **getColumnName()** se o JTable estiver dentro de um JScrollPane;
- Caso contrário será necessário usar de outros artifícios.

Classe TabelaResultados

- Então, a nossa classe TabelaResultados terá um atributo chamado *colunas*, que será um array de String;
- Ela terá o nome das colunas da tabela:

```
package apresentacao;  
  
import javax.swing.table.AbstractTableModel;  
  
import dados.CalculadoraEstatistica;  
  
public class TabelaResultados extends AbstractTableModel {  
    private String[] colunas = { "Sorteado", "Somatorio", "M dia Aritm tica", "M dia  
        Geom trica", "Variancia",  
        "Desvio Padr o", "Amplitude" };  
}
```

Classe TabelaResultados

- Teremos um atributo *calculadora* que será a instância do Singleton da CalculadoraEstatistica;
- Tornamos ela um Singleton para que tanto a tabela de resultados quanto a de valores possa acessar os mesmos dados:

```
private CalculadoraEstatistica calculadora = CalculadoraEstatistica.getInstance();
```

Classe TabelaResultados

- O método **getColumnName()** vai nos retornar o nome da coluna na posição passada como parâmetro, ou seja, o valor na posição do array de Strings:

```
public String getColumnName(int column) {  
    return colunas[column];  
}
```

- Já o método **getColumnCount()** irá nos retornar o tamanho desse array:

```
public int getColumnCount() {  
    return colunas.length;  
}
```

- E o **getRowCount()** vai nos retornar sempre 1, já que só teremos uma valor de média, variância, etc.:

```
public int getRowCount() {  
    return 1;  
}
```

Classe TabelaResultados

- No método **getValueAt()** iremos retornar a chamada de cada método da classe *CalculadoraEstatística*;
- É importante que seja chamado o método de acordo com a sequência descrita no array de Strings *coluna*;
- Faremos uma verificação primeiro, para ver se existem valores na List de inteiros da *CalculadoraEstatística*;
- Caso exista faremos um *switch case* para cada coluna;
- Caso não exista, retornaremos uma String contendo um hífen, indicando que não há nada:

Classe TabelaResultados

```
public Object getValueAt(int linha, int coluna) {  
    if (!calculadora.getValores().isEmpty()) {  
        switch (coluna) {  
            case 0:  
                return calculadora.sortear();  
            case 1:  
                return calculadora.somatorio();  
            case 2:  
                return calculadora.mediaAritmetica();  
            case 3:  
                return calculadora.mediaGeometrica();  
            case 4:  
                return calculadora.variancia();  
            case 5:  
                return calculadora.desvioPadrao();  
            case 6:  
                return calculadora.amplitude();  
        }  
    } else {  
        return " — ";  
    }  
  
    return null;  
}
```

Classe TabelaResultados

- Por último, teremos o método **atualizar()** que irá redesenhar a tabela na tela quando um novo valor for adicionado;
- Esse método apenas chamará o método **fireTableStructureChanged()** da super classe:

```
public void atualizar() {  
    fireTableStructureChanged();  
}
```

- E com isso terminamos a classe;
- Essa classe será o layout de um dos JTables que utilizaremos.

Classe TabelaValores

- Agora vamos implementar a tabela de valores;
- Ela também estenderá a classe AbstractTableModel;
- E utilizará também o Singleton CalculadoraEstatística:

```
package apresentacao;  
  
import javax.swing.table.AbstractTableModel;  
  
import dados.CalculadoraEstatistica;  
  
public class TabelaValores extends AbstractTableModel {
```

- Também terá o atributo colunas, mas como a tabela é única, terá apenas um valor dentro desse array:

```
private String [] colunas = { "Valores" };
```

Classe TabelaValores

- Teremos um atributo *calculadora* que será o Singleton CalculadoraEstatistica:

```
private CalculadoraEstatistica calculadora = CalculadoraEstatistica.getInstance();
```

- Dessa vez o método **getColumnCount()** irá retornar 1, visto que só teremos uma coluna:

```
public int getColumnCount() {  
    return 1;  
}
```

- Já o **getRowCount()** vai retornar o número de valores que possui o List de inteiros da CalculadoraEstatistica:

```
public int getRowCount() {  
    return calculadora.getValores().size();  
}
```


Classe TabelaValores

- O método **getColumnName()** terá a mesma implementação da classe TabelaResultados:

```
public String getColumnName(int column) {  
    return colunas[column];  
}
```

- E o **getValueAt()** irá retornar o valor da List de inteiros da CalculadoraEstatistica de acordo com a linha da tabela:

```
public Object getValueAt(int linha, int coluna) {  
    return calculadora.getValores().get(linha);  
}
```

Classe TabelaValores

- O método **adicionarValor()** vai adicionar uma valor no Singleton da CalculadoraEstatistica e redesenhar a tabela, utilizando o método **fireTableStructureChanged()** da super classe AbstractTableModel:

```
public void adicionaValor(int valor) {  
    calculadora.adicionarValor(valor);  
    fireTableStructureChanged();  
}
```



- E o **limpar()** vai remover todos os valores e redesenhar a tabela seguindo a mesma lógica do **adicionarValor()**:

```
public void limpar() {  
    calculadora.limparValores();  
    fireTableStructureChanged();  
}
```

Instalando Window Builder

- Com isso podemos ir para a última classe, que será o JFrame que abrigará todos a interface gráfica em si;
- Recomenda-se utilizar um plugin para criar o JFrame;
- No Eclipse é necessário baixar o **WindowBuilder**;
- Para baixar o plugin acesse o link:

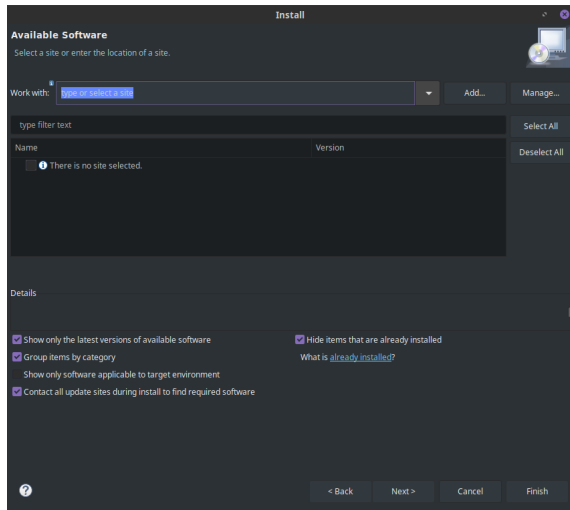
<https://www.eclipse.org/windowbuilder/download.php>

Version	Download and Install		
	Update Site	Zipped Update Site	Marketplace
Latest (1.9.3)	link	link	 Install
Gerrit	link	link	
Last Good Build	link	link	 Install
1.9.3 (Permanent)	link	link	
1.9.2 (Permanent)	link	link	
1.9.1 (Permanent)	link	link	
1.9.0 (Permanent)	link	link	
Archives	link		

- Copie o link da última versão disponível (Latest);

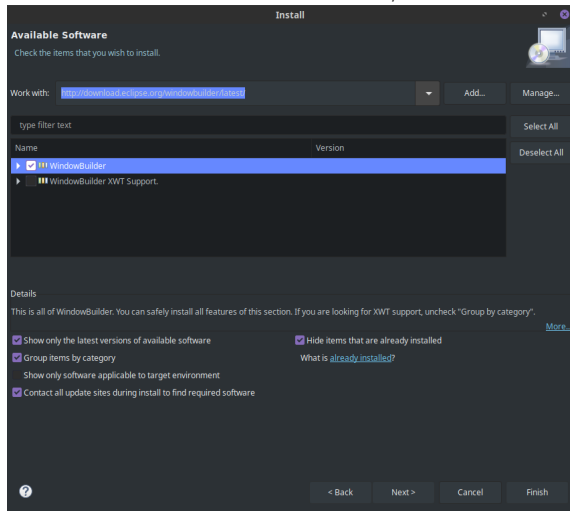
Instalando Window Builder

- No eclipse vá em: **Help > Install New Software**
- E cole no campo **Work With** o link obtido anteriormente:



Instalando Window Builder

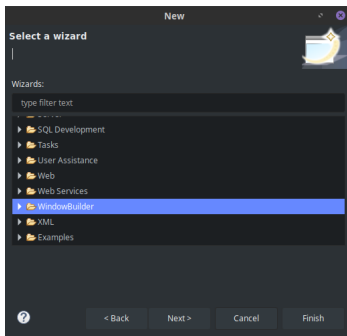
- Pressione ENTER e selecione o **WindowBuilder**;



- Clique em **next** e prossiga com a instalação;

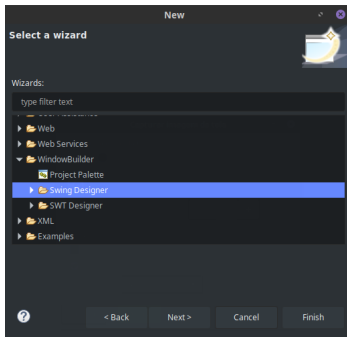
Window Builder

- Agora quando você clicar em **new** estará disponível na opção **other** uma pasta chamada **WindowBuilder**:



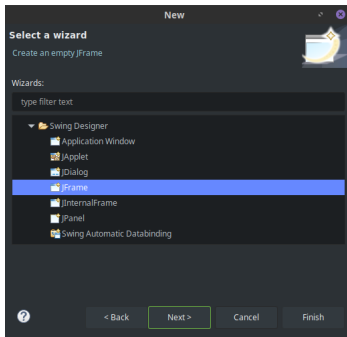
Window Builder

- Utilizaremos os componentes Swing:



Window Builder

- Em nosso exemplo o JFrame



- Mas caso deseje, você pode fazer os componentes gráficos na mão;
- O trabalho é maior e não há tanta necessidade.
- O Netbeans possui recursos próprios para a criação de interfaces gráficas, caso você não queira utilizar o Window Builder;

Classe Calculadora

- Criando a classe Calculadora utilizando o WindowBuilder você tem acesso tanto ao código quanto ao design;
- Assim é possível posicionar os componentes e visualizar o resultado ao mesmo tempo;
- Mas isso não descarta a necessidade de entender o que está sendo feito;
- Por isso, iremos utilizar mais o recurso de código e menos o de design;

Classe Calculadora

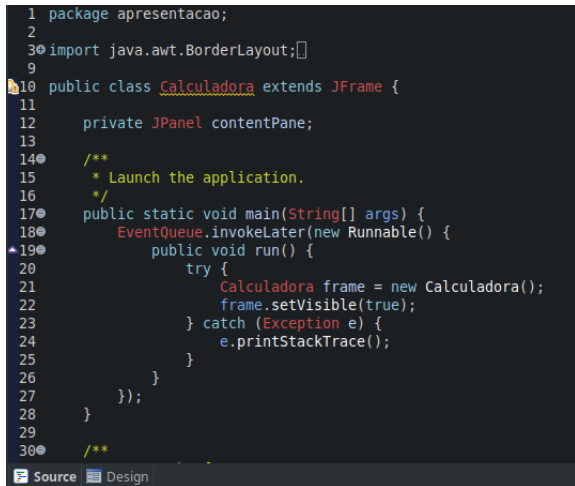
- Observe o que o Window Builder gerou como código-fonte:

```
1 package apresentacao;
2
3 import java.awt.BorderLayout;
4
5
6
7
8
9
10 public class Calculadora extends JFrame {
11
12     private JPanel contentPane;
13
14     /**
15      * Launch the application.
16      */
17     public static void main(String[] args) {
18         EventQueue.invokeLater(new Runnable() {
19             public void run() {
20                 try {
21                     Calculadora frame = new Calculadora();
22                     frame.setVisible(true);
23                 } catch (Exception e) {
24                     e.printStackTrace();
25                 }
26             }
27         });
28     }
29
30     /**
```

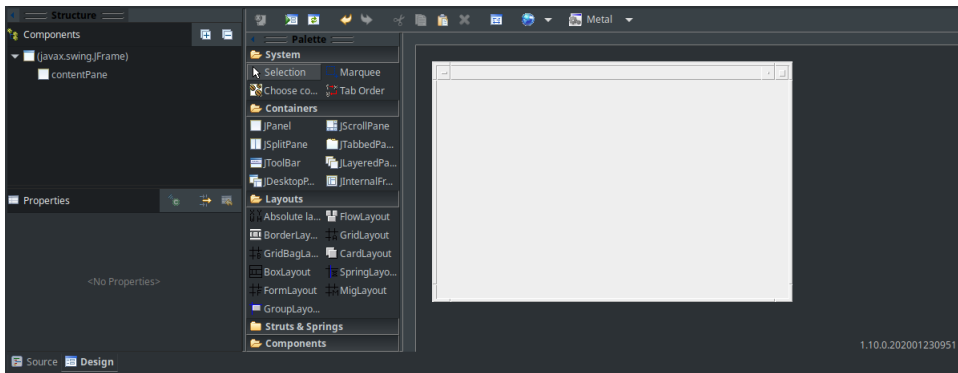
Classe Calculadora

- Clicando na área **Design**, visível no canto inferior esquerdo da imagem, para ver como os componentes serão dispostos na interface gráfica:

```
1 package apresentacao;
2
3 import java.awt.BorderLayout;
4
5
6
7
8
9
10 public class Calculadora extends JFrame {
11
12     private JPanel contentPane;
13
14     /**
15      * Launch the application.
16      */
17     public static void main(String[] args) {
18         EventQueue.invokeLater(new Runnable() {
19             public void run() {
20                 try {
21                     Calculadora frame = new Calculadora();
22                     frame.setVisible(true);
23                 } catch (Exception e) {
24                     e.printStackTrace();
25                 }
26             }
27         });
28     }
29
30     /**
```



Classe Calculadora



- A partir daí fica a seu critério o uso do Window Builder, vamos voltar ao **Source** para explicar a construção do código;

Classe Calculadora

- Caso o Eclipse não consiga encontrar as classes (Ex.: `JPanel cannot be resolved to a type`);
- Vá até o arquivo `module-info.java` que fica dentro da pasta `src` e adicione o comando:

```
module aula13 {  
    requires java.desktop;  
}
```

Classe Calculadora

- Vamos primeiro declarar a classe Calculadora;
- Ela irá estender a classe JFrame;
- E terá um método **main()** dentro dela;
- Que instanciará um objeto do tipo Calculadora e fará a chamada do método **setVisible()** da super classe JFrame;
- Passando **true** como parâmetro;
- Isso fará com que o JFrame fique visível na tela;

```
import javax.swing.JFrame;  
  
public class Calculadora extends JFrame {  
  
    public static void main(String[] args) {  
        Calculadora frame = new Calculadora();  
        frame.setVisible(true);  
    }  
}
```

- Seguiremos uma abordagem um pouco diferente durante a implementação dessa classe;
- Ao invés de uma abordagem linear, como foi feito ao longo de todo o semestre, vamos seguir uma abordagem mais dinâmica;
- Isto é, ao invés de primeiro apresentar todos os atributos e depois todos os métodos, vamos colocar os atributos e métodos conforme utilizarmos.

Classe Calculadora

- Vamos começar alterando alguns parâmetros do JFrame
- Faremos tudo dentro do construtor da classe Calculadora;
- Vamos definir o nome da janela usando o método **setTitle()**, passando uma String como parâmetro:

```
public class Calculadora extends JFrame {  
    public Calculadora() {  
        setTitle("Calculadora Estatística");  
    }  
}
```

Classe Calculadora

- Agora vamos definir a operação de fechar a tela quando pressionar o x do canto superior direito;
- Para isso usamos o método **setDefaultCloseOperation()** passando como parâmetro a constante **JFrame.EXIT_ON_CLOSE** definido pela própria classe **JFrame**:

```
public class Calculadora extends JFrame {  
    public Calculadora() {  
        setTitle("Calculadora Estatística");  
        setDefaultCloseOperation(JFrame.  
            EXIT_ON_CLOSE);  
    }  
}
```

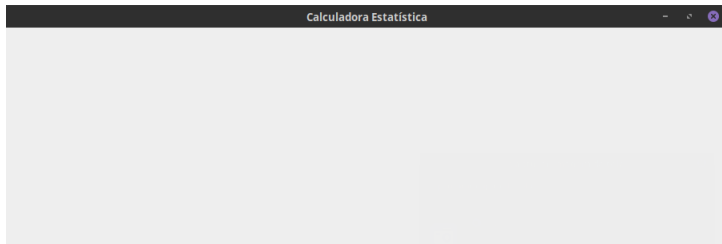
Classe Calculadora

- E definiremos um tamanho para a janela de 900x300;
- Ela irá aparecer 100 pixels a esquerda e a baixo do canto superior esquerdo:
- Faremos isso usando o método **setBound()**;
- Esse método recebe 4 parâmetros:
 - int xInicial;
 - int yInicial;
 - int tamanhoX;
 - int tamanhoY;
- x e y iniciais são opcionais;

```
public class Calculadora extends JFrame {  
    public Calculadora() {  
        setTitle("Calculadora Estatística");  
        setDefaultCloseOperation(JFrame.  
EXIT_ON_CLOSE);  
        setBounds(100, 100, 900, 300);  
    }  
}
```

Classe Calculadora

- Por enquanto temos isso:



Classe Calculadora

- Vamos agora criar um JPanel que irá conter todos os componentes do nosso JFrame;
- Vamos usar o método **setContentPanel()** para atribuir o painel ao JFrame:

```
public class Calculadora extends JFrame {  
    private JPanel painel = new JPanel();  
    public Calculadora() {  
        ...  
        setContentPanel(painel);  
    }  
}
```

Classe Calculadora

- Iremos também definir o layout do JPainel como sendo `null`;
- Para isso vamos usar o método **`setLayout()`** do `JPanel`:
- Isso fará com que os componentes sejam dispostos de acordo com as posições em pixel deles;

```
public class Calculadora extends JFrame {  
    private JPanel painel = new JPanel();  
  
    public Calculadora() {  
        ...  
        setContentPane(painel);  
        painel.setLayout(null);  
    }  
}
```

Classe Calculadora

- Agora vamos criar um outro JPanel que conterá os componentes de entrada de dados;
- Chamaremos ele de painelEntrada;
- Definiremos suas dimensões e setaremos seu layout como `null`;
- E depois adicionaremos ele dentro do JPanel declarado anteriormente:

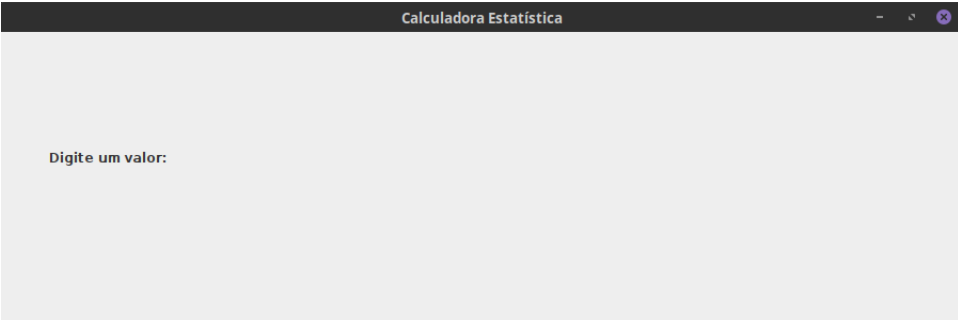
```
public class Calculadora extends JFrame {  
  
    ...  
    private JPanel painelEntrada = new JPanel();  
  
    public Calculadora() {  
  
        ...  
        painelEntrada.setBounds(15, 80, 280, 173);  
        painelEntrada.setLayout(null);  
        painel.add(painelEntrada);  
  
    }  
  
}
```

Classe Calculadora

- Vamos criar um JLabel informando o que o usuário precisa digitar;
- Dentro do construtor vamos definir suas dimensões e adicionar ele dentro do painel de entrada de dados:

```
public class Calculadora extends JFrame {  
  
    ...  
    JLabel infoCaixaTexto = new JLabel("Digite um  
    valor:");  
  
    public Calculadora() {  
  
        ...  
        infoCaixaTexto.setBounds(30, 30, 200, 15);  
        painelEntrada.add(infoCaixaTexto);  
  
    }  
  
}
```


Classe Calculadora



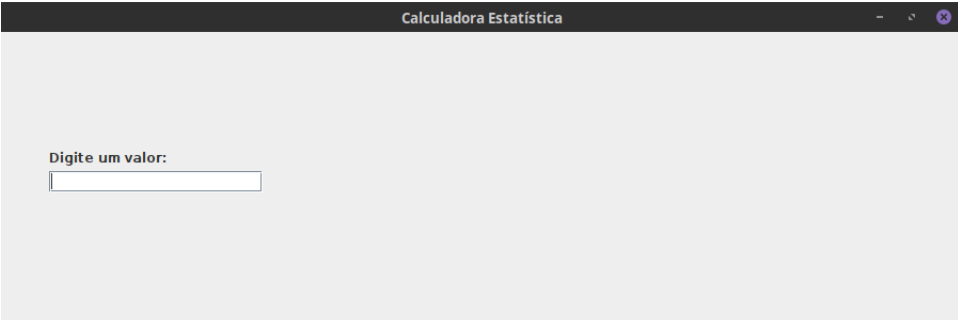
A screenshot of a Java Swing window titled "Calculadora Estatística". The window has a dark gray title bar with standard OS controls (minimize, maximize, close) on the right. The main content area is light gray and contains a single text input field with the placeholder text "Digite um valor:".

Classe Calculadora

- Agora vamos criar um JTextField;
- Ele servirá como campo para a digitação de dados pelo usuário;
- Lembre-se que a entrada de dados acontece na forma de String!

```
public class Calculadora extends JFrame {  
    ...  
    private JTextField caixaTexto = new JTextField()  
    ;  
    public Calculadora() {  
        ...  
        caixaTexto.setBounds(30, 50, 200, 20);  
        painelEntrada.add(caixaTexto);  
    }  
}
```

Classe Calculadora



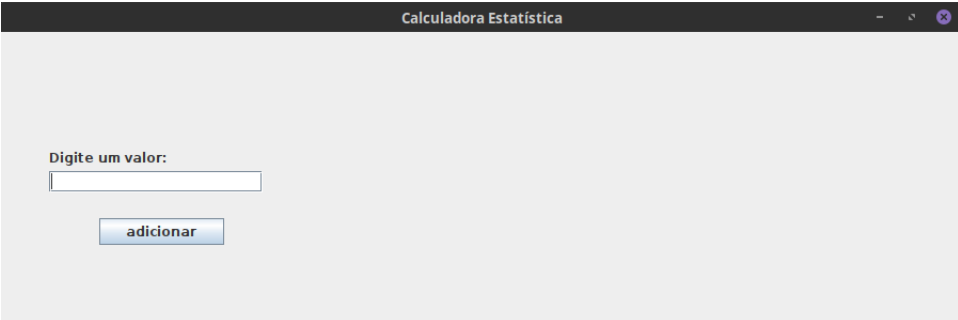
A screenshot of a Java Swing window titled "Calculadora Estatística". The window has a dark gray title bar with standard OS controls (minimize, maximize, close) on the right. The main content area is light gray. On the left side, there is a label "Digite um valor:" followed by a single-line text input field.

Classe Calculadora

- Agora vamos criar os botões;
- O primeiro deles é o de adicionar;
- Criaremos o JButton passando o texto do botão como parâmetro no construtor;
- Depois setaremos as suas dimensões com o **setBounds()**;
- E adicionaremos ele ao painel de entrada de dados;
- Mais tarde adicionaremos a ação do botão!

```
public class Calculadora extends JFrame {  
  
    ...  
    private JButton botaoAdicionar = new JButton("adicionar");  
  
    public Calculadora() {  
  
        ...  
        botaoAdicionar.setBounds(77, 94, 117, 25);  
        painelEntrada.add(botaoAdicionar);  
  
    }  
}
```

Classe Calculadora



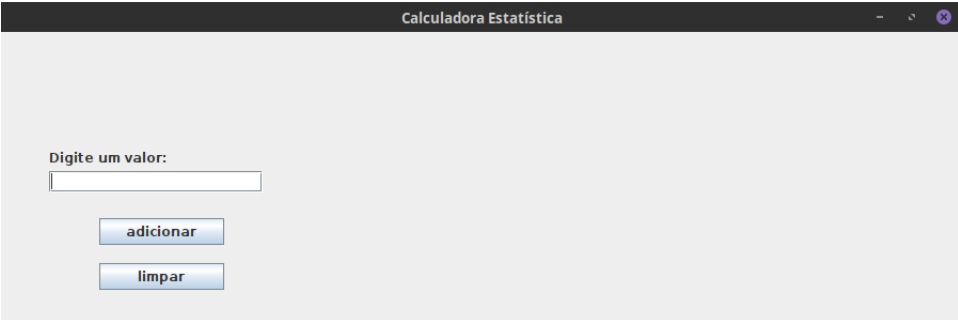
The image shows a screenshot of a Java Swing window titled "Calculadora Estatística". The window has a dark gray title bar with standard OS controls (minimize, maximize, close) on the right. The main content area is light gray. On the left side, there is a text label "Digite um valor:" followed by a text input field. Below the input field is a button labeled "adicionar".

Classe Calculadora

- Seguiremos a mesma lógica para o botão de limpar;
- Passaremos o texto como parâmetro na sua instanciação;
- No construtor, definiremos a sua dimensão e o adicionaremos ao painel de entrada de dados;
- Novamente, a ação do botão será implementada mais tarde;

```
public class Calculadora extends JFrame {  
  
    ...  
    private JButton botaoLimpar = new JButton("limpar");  
  
    public Calculadora() {  
  
        ...  
        botaoLimpar.setBounds(77, 136, 117, 25);  
        painelEntrada.add(botaoLimpar);  
  
    }  
}
```

Classe Calculadora



The image shows a Java Swing window titled "Calculadora Estatística". The window has a dark gray title bar with standard OS controls (minimize, maximize, close). The main content area is light gray. On the left side, there is a text label "Digite um valor:" followed by a text input field. Below the input field are two buttons: "adicionar" and "limpar".

Calculadora Estatística

Digite um valor:

adicionar

limpar

Classe Calculadora

- Agora vamos criar um JScrollPane para abrigar a nossa tabela;
- Assim como mencionado anteriormente, o JTable só exibe o título quando dentro de um JScrollPane!
- Daremos o nome do JScrollPane de painelScrollTabelaResultados:
- Assim como feito para os outros componentes, vamos definir suas dimensões e adicioná-lo ao painel, no caso, ao painel do JFrame;

```
public class Calculadora extends JFrame {  
  
    ...  
    private JScrollPane painelScrollTabelaResultados  
        = new JScrollPane();  
  
    public Calculadora() {  
  
        ...  
        painelScrollTabelaResultados.setBounds(10,  
        10, 880, 50);  
        painel.add(painelScrollTabelaResultados);  
  
    }  
  
}
```


Classe Calculadora

- Criado o JScrollPane, podemos criar a tabela que ficará dentro dele;
- Será um JTable e dentro do construtor passaremos uma instância da nossa classe TabelaResultados;
- Depois vamos usar o método **setViewportView()** do JScrollPane para colocar a tabela dentro do painel com scroll:

```
public class Calculadora extends JFrame {  
  
    ...  
    private JTable tabelaResultados;  
    private TabelaResultados resultados = new  
        TabelaResultados();  
  
    public Calculadora() {  
  
        ...  
        tabelaResultados = new JTable(resultados);  
        painelScrollTabelaResultados.setViewportView(  
            tabelaResultados);  
    }  
}
```

Classe Calculadora

Calculadora Estatística

Sorteado	Somatorio	Média Aritmética	Média Geométrica	Variancia	Desvio Padrão	Amplitude
-	-	-	-	-	-	-

Digite um valor:

adicionar

limpar

Classe Calculadora

- Faremos exatamente o mesmo processo para tabela de valores;
- Criaremos um JScrollPane para abrigar a tabela;
- E adicionaremos ao JFrame:

```
public class Calculadora extends JFrame {  
  
    ...  
    private JScrollPane paineScrollTabelaValores =  
        new JScrollPane();  
  
    public Calculadora() {  
  
        ...  
        paineScrollTabelaValores.setBounds(307, 80,  
            173, 173);  
        painel.add(paineScrollTabelaValores);  
    }  
}
```

Classe Calculadora

- Criaremos o JTable e um objeto da classe TabelaValores;
- Passaremos o objeto da TabelaValores no construtor da JTable e adicionaremos o JTable ao JScrollPane;

```
public class Calculadora extends JFrame {  
    ...  
    private JTable tabelaValores;  
    private TabelaValores valores = new TabelaValores();  
  
    public Calculadora() {  
        ...  
        tabelaValores = new JTable(valores);  
        paineScrollTabelaValores.setViewportViewView(  
            tabelaValores);  
    }  
}
```

Classe Calculadora

Calculadora Estatística

Sorteado	Somatorio	Média Aritmética	Média Geométrica	Variancia	Desvio Padrão	Amplitude
-	-	-	-	-	-	-

Digite um valor:

adicionar

limpar

Valores

Classe Calculadora

- Por fim, vamos implementar as ações dos botões;
- Vamos começar pelo botão de adicionar;
- A classe JButton possui um método **addActionListener()** que recebe um objeto do tipo ActionListener;
- Tal objeto, como o nome já diz, é um "ouvinte" de ação;
- Isto é, quando pressionado vai realizar algo;
- Para isso vamos criar uma classe Anônima;
- Uma classe anônima é uma classe que é criada no momento da instanciação de um objeto do seu tipo;
- Não faz sentido criar uma classe só para uma ação de um botão sendo que não vamos criar dois botões que realizam a mesma função!

Classe Calculadora

- Mas caso queira, faça uma classe separada;
- Vamos começar chamando o método **addActionListener()** do botão de adicionar;

```
botaoAdicionar.addActionListener (...);
```

- Na declaração dos parâmetros vamos instanciar um novo ActionListener e implementá-lo, mas o implementaremos após a declaração dele;
- A classe Abstrata ActionListener pede a implementação do método **public void actionPerformed(ActionEvent arg0):**

```
botaoAdicionar.addActionListener( new ActionListener(){  
    public void actionPerformed(ActionEvent arg0) {  
        // Acao realizada pelo botao  
    }  
});
```

Classe Calculadora

- Esse método (actionPerformed) será executado toda vez que o botão for pressionado.
- No caso do botão de adicionar, vamos primeiro pegar o valor que o usuário digitou na caixa de texto;
- Usaremos o método **getText()** da JTextField para obter o valor digitado pelo usuário;
- Depois, vamos converter esse valor para um inteiro;
- Por fim, vamos usar o método **adicionaValor()** da classe TabelaValores que criamos anteriormente;

```
botaoAdicionar.addActionListener( new ActionListener() {  
    public void actionPerformed(ActionEvent arg0) {  
        valores.adicionaValor(Integer.parseInt(caixaTexto.getText()))/  
    }  
});
```


Classe Calculadora

- Depois disso vamos atualizar as tabelas;
- O método **adicionaValor()** já atualiza automaticamente a tabela (implementamos ele para que fosse assim);
- Mas a tabela de resultados precisa ser atualizar manualmente;
- Faremos isso chamando o método **atualizar()** dela, que a partir da CalculadoraEstatistica recalcula os resultado:

```
botaoAdicionar.addActionListener( new ActionListener() {  
    public void actionPerformed(ActionEvent arg0) {  
        valores.adicionaValor(Integer.parseInt(caixaTexto.getText()));  
        resultados.atualizar();  
    }  
});
```

- Por fim vamos usar o método **setText()** do JTextField para “limpar” o texto e preparar a caixa de texto para o usuário poder repetir a operação;
- Para isso vamos passar uma String vazia como parâmetro:

```
botaoAdicionar.addActionListener( new ActionListener() {  
    public void actionPerformed(ActionEvent arg0) {  
        valores.adicionaValor(Integer.parseInt(caixaTexto.getText()));  
        resultados.atualizar();  
        caixaTexto.setText("");  
    }  
});
```

Classe Calculadora

- A implementação do botão de limpar segue o mesmo princípio;
- Entretanto, ele chama o método **limpar()** da classe TabelaValores que limpa a *List* de inteiros da CalculadoraEstatistica e atualiza a tabela;
- Após isso atualiza a tabela de resultados também;

```
botaoLimpar.addActionListener( new ActionListener() {  
    public void actionPerformed(ActionEvent arg0) {  
        valores.limpar();  
        resultados.atualizar();  
    }  
});
```

- Feito isso a interface gráfica está completa;
- Novamente é importante ressaltar um detalhe com relação ao uso do JTable;
- Caso você não utilize ele dentro de um JScrollPane, você precisará realizar algumas mudanças na implementação da classe que estende a AbstractTableModel:
 1. No método **getRowCount()** retorne o tamanho da lista a ser exibido -1;
 2. No método **getValueAt()**, verifique se a linha é igual a 1, caso sim, retorne o nome da coluna;
 3. Caso a linha seja diferente de 1, retorne o objeto na posição *linha - 1* da lista de objetos a ser exibido.
- O código-fonte está disponível nesse [link](#).

Seções

Exemplo

Resolução

Exercício

Exercício


- Escolha três das classes criadas na Aula Prática 6: Classes Abstratas que estendam a classe abstrata Gerador e traga-as para o projeto;
- Crie três botões e uma caixa de texto na interface gráfica do exemplo anterior para fazer a utilização de cada uma das classes trazidas;
 - A caixa de texto fará a leitura da quantidade de valores que o usuário quer gerar e os botões irá gerar tal quantidade;
 - Os valores gerados devem ser adicionados a classe TabelaValores;
 - Exemplo:

Sorteado	Somatorio	Média Aritmética	Média Geométrica	Variância	Desvio Padrão	Amplitude
-	-	-	-	-	-	-

Digite um valor:

Valores

Gerar Valores

 KUWAKI, V. T. F. Modelo de slides udesc lattex. In: . [S.l.]: Disponível em: <<https://github.com/takeofriedrich/slidesUdescLattex>>. Acesso em: 24 jan. 2020.

Duvidas:
Vinicius Takeo Friedrich Kuwaki
vinicius.kuwaki@edu.udesc.br
github.com/takeofriedrich