

# Interface Gráfica

em Java - Aula 2

**Vinicius Takeo Friedrich Kuwaki**  
Universidade do Estado de Santa Catarina

# Seções

Exemplo

Resolução

Exercício

## Exemplo

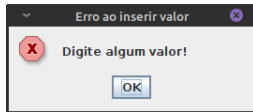
- O propósito deste exercício é apresentar alguns outros componentes e eventos que podem ser utilizados na interface gráfica;
- Portanto, continuando o exercício da aula anterior:
  - Substitua os três botões por uma JComboBox e um único botão:

The screenshot shows a Java Swing window titled "Calculadora Estatística". At the top, there is a table with the following columns: Sorteado, Somatorio, Média Aritmética, Média Geométrica, Variância, Desvio Padrão, and Amplitude. Below the table, there are input fields and buttons. On the left, there is a label "Digite um valor:" followed by a text input field, and two buttons labeled "adicionar" and "limpar". In the center, there is a large empty rectangular box labeled "Valores". On the right, there is another label "Digite um valor:" followed by a text input field, a JComboBox with "Fibonacci" selected, and a button labeled "gerar".

Exemplo de JComboBox para gerenciar as classes Abstratas

## Exemplo

- Além disso, mostre as seguintes exceções:
  - Usuário deixou a caixa em branco e clicou em adicionar;
  - Usuário clicou em limpar e não havia valores na lista;
  - Usuário clicou em gerar e a caixa estava em branco;
  - Usuário entrou com valores incorretos (não digitou inteiros);
- Para isso, utilize o JOptionPane:



Exemplo de JOptionPane

# Seções

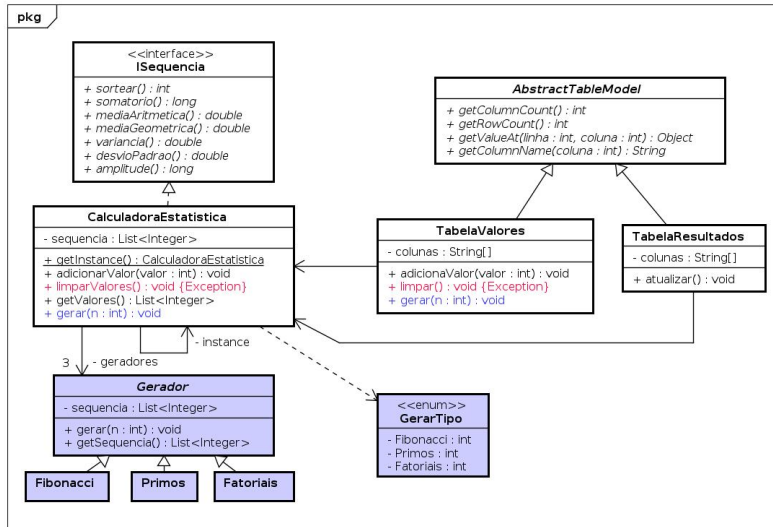
Exemplo

Resolução

Exercício

- Para melhor organizar o back-end, vamos alterar apenas os itens destacados no diagrama a seguir;
- As classes e métodos destacados em azul serão criadas (ou buscadas de projetos anteriores);
- Já os métodos em vermelho serão apenas alterados;

# Resolução - Diagrama



## Resolução - Gerador e classes filhas

- Vamos começar buscando as classes que foram utilizada na Aula Prática 6 (Gerador, Fibonacci, Primos e Fatoriais);
- Para aproveitarmos a reutilização de código, vamos copiar e colar os arquivos com os códigos-fontes dentro do projeto. As classes se encontram nesse [link](#).



## Resolução - Enum GerarTipo

- Vamos criar um Enum para enumerarmos as opções que utilizaremos no JComboBox.
- Cada item enumerado será responsável por utilizar o método **gerar()** de uma das especializações da superclasse Gerador.

```
public enum GerarTipo {  
    FIBONACCI, PRIMOS, FATORIAL;  
}
```

## Resolução - Classe CalculadoraEstatística

- No fonte da classe CalculadoraEstatística vamos tratar o primeiro evento que pode ocorrer ao pressionar um dos botões da interface;
- Caso o usuário use o método **limpar()** e a lista esteja vazia, vamos lançar uma exceção:

```
public void limparValores() throws Exception {  
    if (sequencia.size() > 0) {  
        this.sequencia.clear();  
    } else {  
        throw new Exception("Lista vazia!");  
    }  
}
```

## Resolução - Classe CalculadoraEstatística

- Agora vamos adicionar um array de geradores, onde cada posição corresponde a uma instância de uma das classes filhas de gerador;

```
public class CalculadoraEstatistica {  
    ...  
    private Gerador geradores[] = new Gerador[] { new Fibonacci(), new NumerosPrimos()  
    }, new Fatoriais() };  
    ...  
}
```

- Agora, para cada enumerado, o método **gerar()** vai escolher o qual das classes filhas do Gerador ela utilizará o método **gerar()**;
- Para isso, tal método vai receber uma das opções do enum e a quantidade de valores a serem gerados;
- Utilizaremos um switch case para cada opção do enum GerarTipo e utilizaremos o método **getSequencia()** da super classe Gerador para obter os valores gerados.

```
public void gerar(GerarTipo tipo, int n) {  
    switch (tipo) {  
        case FATORIAL:  
            geradores[0]. gerar(n);  
            this.sequencia.addAll(geradores[0].getSequencia());  
            break;  
        case FIBONACCI:  
            geradores[1]. gerar(n);  
            this.sequencia.addAll(geradores[1].getSequencia());  
            break;  
        case PRIMOS:  
            geradores[2]. gerar(n);  
            this.sequencia.addAll(geradores[2].getSequencia());  
            break;  
    }  
}
```

## Resolução - Classe TabelaValores

- Terminadas as modificações nas camadas de baixo, vamos modificar as classes da camada de apresentação;
- A primeira delas é a classe TabelaValores;
- Como a interface GUI chama os métodos dessa classe, criaremos um método **gerar()** ali também.
- Esse método apenas chama o método **gerar()** da classe CalculadoraEstatística e atualiza a tabela na GUI.

```
public void gerar(GerarTipo tipo, int n) {  
    calculadora.gerar(tipo, n);  
    fireTableStructureChanged();  
}
```

- Vamos modificar o método **limpar()** para que ele possa repassar a exceção para a próxima classe:

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

- E por fim, vamos criar os componentes Swing que realizam de fato as funcionalidades:

## Resolução - Classe Calculadora

- Vamos criar um JPanel onde colocaremos o JTextField, o JComboBox e o JButton;
- Nesse exemplo, vamos utilizar uma borda ao redor do painel;
- A biblioteca javax.swing possui um Factory feito especialmente para criar bordas;
- Utilizaremos uma borda de título:

```
public Calculadora() {  
    ... // Continuacao do codigo da aula anterior  
  
    JPanel painelGerador = new JPanel();  
    painelGerador.setLayout( null );  
    painelGerador.setSize( 200, 173 );  
    painelGerador.setLocation( 500, 80 );  
    painelGerador.setBorder( javax.swing.BorderFactory.createTitledBorder( "Gerar  
Valores" ) );  
    add( painelGerador );  
}
```



## Resolução - Classe Calculadora

- Também colocaremos um JTextField para realizar a entrada de dados:

```
public Calculadora() {  
    ... // Ainda no construtor da classe Calculadora  
  
    JTextField valorGerador = new JTextField();  
    valorGerador.setBounds(25, 35, 150, 20);  
    painelGerador.add(valorGerador);  
}
```

- E agora, vamos criar a JComboBox;
- Ela será tipada de acordo com os valores do enum GerarTipo, declarado anteriormente;
- Vamos utilizar o método **values()** do enum;
- A JComboBox é tipada com algum objeto T, utilizaremos o próprio enum como T:

- Vamos passar um array no construtor da JComboBox:

```
public Calculadora() {  
    ... // Ainda no construtor da classe Calculadora  
  
    JComboBox<GerarTipo> comboBox = new JComboBox<GerarTipo>(GerarTipo.values());  
    comboBox.setBounds(25, 80, 150, 20);  
    painelGerador.add(comboBox);  
}
```

- Por fim, vamos criar o JButton que contemplará a ação de gerar os valores de acordo com o valor selecionado na JComboBox:

## Resolução - Classe Calculadora

```
public Calculadora() {  
    ... // Ainda no construtor da classe Calculadora  
  
    JButton botaoGerador = new JButton("gerar");  
    botaoGerador.setBounds(50, 115, 100, 20);  
    painelGerador.add(botaoGerador);  
}
```

- Dentro da ação desse botão vamos tratar dois eventos ao mesmo tempo;
- Caso o usuário deixe a caixa em branco ao pressionar o botão;
- Ou, ainda, caso o usuário digite um valor que não seja inteiro;
- Faremos isso, utilizando a exceção `NumberFormatException` que o método `getText()` do `JTextField` lança.

- Vamos chamar o método **gerar()** da classe TabelaValores passando o enum que o usuário escolheu no JComboBox;
  - Para isso, utilizamos o método **getSelectedItem()**;
- Também passaremos o valor que o usuário digitou na caixa de texto;
- Por fim, chamamos o método **atualiza()** da TabelaResultados;
- Tudo isso dentro do bloco **try** para podermos capturar uma eventual exceção lançada pelo **getText()**:

## Resolução - Classe Calculadora

```
public Calculadora() {  
    ... // Ainda no construtor da classe Calculadora  
  
    botaoGerador.addActionListener(new ActionListener() {  
        public void actionPerformed(ActionEvent arg0) {  
            try {  
                valores.gerar((GerarTipo) comboBox.getSelectedItem(), Integer.  
parseInt(valorGerador.getText()));  
                resultados.atualizar();  
                valorGerador.setText("");  
            } catch (NumberFormatException e) {  
                ...  
            }  
        }  
    });  
}
```

- Note que precisamos fazer um Casting na chamada do método **getSelectedItem()** do JComboBox, pois ele retorna um Object;

- Agora vamos exibir um JOptionPane para a exceção;
- Utilizaremos o método estático da classe JOptionPane chamado **showMessageDialog()**,
- Esse método requer quatro parâmetros:
  - O componente (em nosso caso null porque ele criará outro);
  - A mensagem e o título (ambas strings);
  - E o tipo enumerado de JOptionPane:
    - ERROR\_MESSAGE; (vamos usar esse)
    - INFORMATION\_MESSAGE;
    - QUESTION\_MESSAGE;
    - etc... (Veja a documentação)

## Resolução - Classe Calculadora

```
public Calculadora() {  
    ... // Ainda no construtor da classe Calculadora  
    botaoGerador.addActionListener(new ActionListener() {  
        public void actionPerformed(ActionEvent arg0) {  
            try {  
                ...  
            } catch (NumberFormatException e) {  
                JOptionPane.showMessageDialog(null, "Digite uma entrada valida!", "  
                Erro ao ler valor",  
                JOptionPane.ERROR_MESSAGE);  
            }  
        }  
    });  
}
```

- Terminado o tratamento de erro do botão gerar, vamos para o botão limpar, que também irá utilizar um `try/catch`;
- Basta envolver todo o código criado na aula anterior dessa função com um `try` e adicionar a cláusula `catch` com o `JOptionPane`;
- A lógica é a mesma do botão gerar:



## Resolução - Classe Calculadora

```
public Calculadora() {  
    ... // Ainda no construtor da classe Calculadora  
  
    botaoLimpar.addActionListener(new ActionListener() {  
        public void actionPerformed(ActionEvent arg0) {  
            try {  
                valores.limpar();  
                resultados.atualizar();  
            } catch (Exception e) {  
                JOptionPane.showMessageDialog(null, e.getMessage(), "Erro ao limpar  
valores",  
                JOptionPane.ERROR_MESSAGE);  
            }  
        }  
    });  
}
```

## Resolução - Classe Calculadora

- Por fim, vamos fazer o tratamento de erro do botão adicionar;
- Nele, envolveremos o código da função criado na aula anterior com outro `try`, que irá capturar a exceção `NumberFormatException`;

```
public Calculadora() {  
    ... // Ainda no construtor da classe Calculadora  
    botaoAdicionar.addActionListener(new ActionListener() {  
        public void actionPerformed(ActionEvent arg0) {  
            try {  
                valores.adicionaValor(Integer.parseInt(caixaTexto.getText()));  
                resultados.atualizar();  
                caixaTexto.setText("");  
            } catch (NumberFormatException e) {  
                JOptionPane.showMessageDialog(null, "Digite algum valor!", "Erro ao  
inserir valor", JOptionPane.ERROR_MESSAGE);  
            }  
        }  
    });  
    ...  
}
```

- O código-fonte se encontra nesse [link](#).

# Seções


Exemplo

Resolução

Exercício

## Exercício

- Crie uma interface gráfica para o projeto da Lista de Contatos da Aula Prática 8: Tratamento de Exceções;
- A interface deve listar os contatos do arquivo em forma de tabela;
- A interface também deve permitir o cadastro de novos contatos;
- Cada exceção lançada pelo sistema deve ser exibida em um JOptionPane informando a mensagem do erro.

 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](mailto:vinicius.kuwaki@edu.udesc.br)  
[github.com/takeofriedrich](https://github.com/takeofriedrich)



**UDESC**  
UNIVERSIDADE  
DO ESTADO DE  
SANTA CATARINA