



# Programação Orientada a Objetos

## Introdução à Java

**Tércio de Moraes**

<sup>1</sup>Ciência da Computação  
Campus Arapiraca  
Universidade Federal de Alagoas – UFAL

Arapiraca, 28 de julho de 2024





# Agenda

- 1 Introdução à sintaxe Java
- 2 Memória
- 3 Estrutura de decisão
- 4 Estrutura de repetição
- 5 Funções
- 6 Entrada e saída de dados



# Elementos da Linguagem

## Definindo Java

➤ **Coleção de objetos** que se comunicam através de **invocação de métodos** (funções)

## Elementos da linguagem

- **Objeto** – elemento central de OO, composto por bluevariáveis e **funções**.
- **Classe** – é o **modelo do objeto**. Equivalente ao tipo de dado.
- **Variáveis de instância** – conjunto de variáveis que **pertencem a um objeto**
- **Métodos** – **implementa o comportamento do objeto**. Em outras palavras, é o **papel** que o objeto desempenha.



# Estrutura de um código Java

```
1 package code.java;  
2  
3 import java.util.Vector;  
4 import java.util.concurrent.ForkJoinPool;  
5  
6 public class Pessoa {  
7     private String nome = null;  
8     private String telefone = null;  
9     private String email = null;  
10  
11     public String getNome() {  
12         return this.nome;  
13     }  
14     ...  
15  
16     public void setEmail(String email) {  
17         if(email != null && email.contains("@"))  
18             this.email = email;  
19     }  
}
```

**Namespace para classes e interfaces**

**Importação de bibliotecas e classes**

**Declaração de variáveis de instância**

**Implementação de funções**



# Sintaxe Básica em Java

## Características

- **Case sensitive** – Java **difere letras maiúsculas de minúsculas**
- **Nome de classe** – Palavras iniciadas com **letras maiúsculas** (opcional)
- **Nome de arquivo de classe** – Deve ser **o mesmo nome da classe implementada** (obrigatório). Exemplo: `HelloWorld.java`
- **Nome de método** – Primeira palavra **iniciada com letra minúsculas**. Palavras restantes **iniciada com letra maiúscula**. Exemplo: `atualizarDadosCliente(...)`
- **Comando** – Todo comando é finalizado com um **ponto e vírgula**



# Sintaxe Básica em Java

## Características

➤ **Case sensitive**

➤ **Nomes de variáveis**

➤ **Nomes de métodos**

(obrigatoriamente

➤ **Nomes de classes**

```
1 public class HelloWorld{  
2     public static void main(string args[]){  
3         System.out.println("Olá, mundo!");  
4     }  
5 }
```

restantes **iniciada com letra maiúscula**. Exemplo: `atualizarDadosCliente(...)`

➤ **Comando** – Todo comando é finalizado com um **ponto e vírgula**



# Modelo de Memória: Variáveis Primitivas

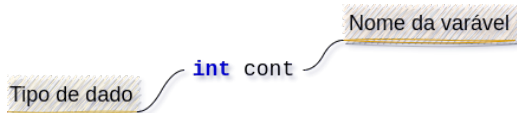
## Variáveis primitivas

- Dados são **fortemente tipificados**
- É **obrigatório** definir o tipo de dado
- São 8 tipos primitivos em Java
  - **Inteiros**: **byte** (1 byte), **short** (2 bytes), **int** (4 bytes), **long** (8 bytes)
  - **Ponto flutuante**: **float** (4 bytes), **double** (8 bytes)
  - **Textual**: **char** (2 bytes) – UNICODE
  - **Lógico**: **boolean** (1 bit)
- Qual a **vantagem** da tipificação forte?
  - Erros podem ser facilmente identificados pelo compilador
  - Gerenciamento de memória
  - Diminui erros durante a programação



# Manipulação de Variáveis

- Como visto antes, devemos definir o **tipo da variável**
- **Atribuição de valores** segue a mesma ideia de outras linguagens

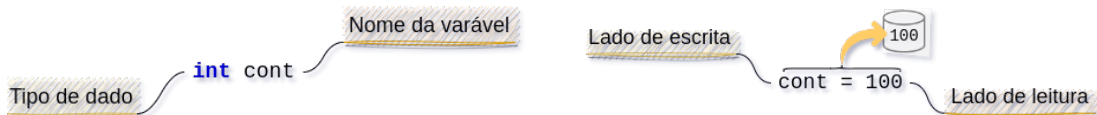






# Manipulação de Variáveis

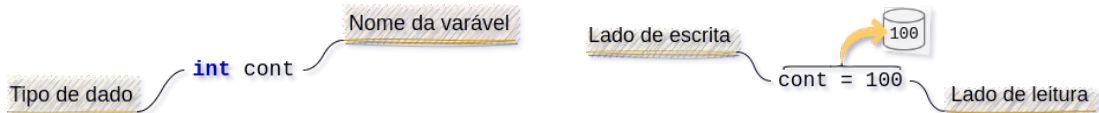
- Como visto antes, devemos definir o **tipo da variável**
- Atribuição de valores** segue a mesma ideia de outras linguagens



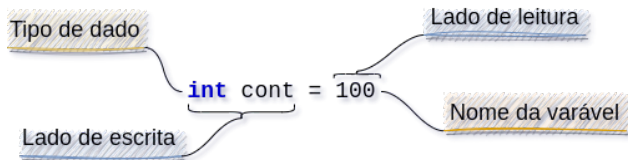


## Manipulação de Variáveis

- Como visto antes, devemos definir o **tipo da variável**
- Atribuição de valores** segue a mesma ideia de outras linguagens



- Juntando tudo ...





## Outros Exemplos sobre Variáveis

➡ Variações para trabalhar com variáveis

```
1 pi          = 3.141f;  
2 foo         = 20.32d;  
3 caractere   = 'v';
```

➡ Outras variações

```
1 int a=2, b=4, c=6;  
2 float pi  = 3.14f;  
3 double de = 0.22d;  
4 char a    = 'v';
```

### Tipificação na prática

➡ Considere o seguinte código

```
1 public class Tipagem {  
2     public static void main(String args[]){  
3         int    x1    = 0;  
4         float  x2    = 3;  
5         x2 = x1; // O que acontece?  
6         x1 = x2; // O que acontece?  
7         System.out.print("x1=" + x1  
8                             + ", x2=" + x2);  
9     }  
10 }
```



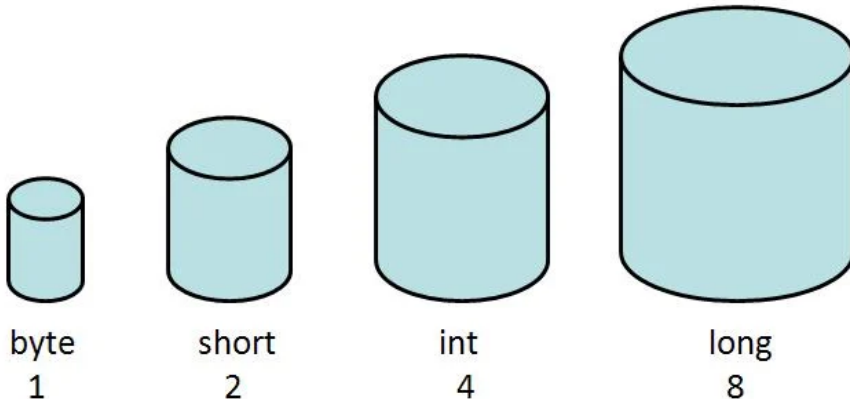
## Type Casting

### Voltando ao erro anterior

- O erro ocorre por causa do **espaço de memória** reservado para o tipo de variável.
- São dois casos:
  - Atribuição de uma **“memória menor”** para uma **“memória maior”**
    - Ocorre de forma implícita
  - Atribuição de uma **“memória maior”** para uma **“memória menor”**
    - Deve ser feito explicitamente
    - Operador **type cast** indicando entre parênteses para qual tipo que deseja converter
    - Em nosso exemplo: `x1 = (int)x2`
- **Atenção:** Os dados serão lidos seguindo o **tipo de dado da variável**



## Type Casting





## Modelo de Memória: Variáveis Derivadas

### Conhecidas como variáveis compostas

- Tipos de dados formados pela **composição** de outros **tipos de dados**
- Podem ser
  - **Homogêneas** – conjunto de valores de um **mesmo tipo de dado** (vetoriais)
  - **Heterogênea** – conjunto de valores de **diferentes tipos de dados** (estruturais, registros)

### O mais popular: vetores unidimensionais

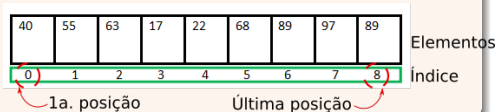
- Conhecidas como **Array**
- **Python** abstrai vetores em um único tipo: **lista**
- Cada **elemento** de um **array** é endereçado com um **índice numérico** iniciado por **0**



## Modelo de Memória: Array na Prática

### Declarando um array

```
<tipo_dado>[] <nome_variavel>;  
<tipo_dado> <nome_variavel>[];  
  
int[] arrayInteiros;  
int arrayInteiros[];
```



### Construindo um array

```
Apesar de declarado, o array não tem espaço de memória alocado  
  
<nome_variavel> = new <tipo_dado>[<tamanho_array>];  
arrayInteiros = new int[9]; OU int[] arrayInteiros = new int[9];  
  
Outra forma:  
arrayInteiros = {40,55,63,17,22,68,89,97,89};
```



## Modelo de Memória: Array na Prática

### Localização de um elemento no array

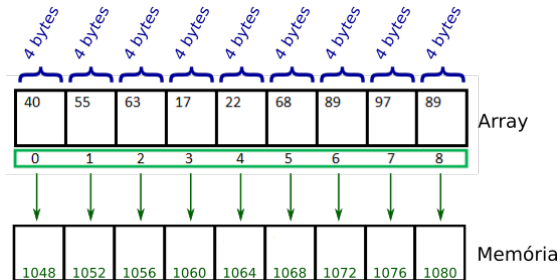
- O elemento do **array** é identificado pelo seu índice: **arrayInteiro[0]**
- Os índices têm uma distância entre si do **tamanho do tipo de dado** armazenado

### Escrevendo no array

```
arrayInteiro[0] = 40;
```

### Lendo do array

```
int i = arrayInteiro[0];
```







## Praticando Arrays em Java

🔧 Programa que escreve e lê em um array

```
1 public class ArrayDemo{
2     public static void main (String[] args){
3         int[] demo = new int[5];
4         demo[0] = 96;           //escrita
5         demo[1] = -35;          //escrita
6         demo[2] = demo[0] + demo[1]; //leitura e escrita
7         demo[3] = demo[0] - demo[1]; //leitura e escrita
8         demo[4] = demo[3] * demo[1]; //leitura e escrita
9         System.out.println(demo[0] + ", " + demo[1] + ", "
10                            + demo[2] + ", " + demo[3] + ", "
11                            + demo[4]);
12     }
13 }
```



## Praticando Arrays em Java

🔧 Programa que escreve e lê em um array

```
1 public class ArrayDemo{
2     public static void main(String[] args)
3     {
4         int[] demo = new int[5];
5         demo[0] = 1; //escrita
6         demo[1] = 3; //escrita
7         demo[2] = demo[0] + demo[1]; //leitura e escrita
8         demo[3] = demo[1]; //leitura e escrita
9         demo[4] = demo[3] * demo[1]; //leitura e escrita
10        System.out.println(demo[0] + " + " + demo[1] + " = "
11                               + demo[2] + " + " + demo[3] + " = "
12                               + demo[4]);
13    }
```



## Modelo de Memória: Arrays Multidimensionais

Da lista para a tabela

- Também chamado de **matriz**
- Composto por um **conjunto de arrays**

### Em Java

- Similar ao array
- `int twoDim[] [] = new int[4][5];`
- Escrita: `twoDim[0][0] = 8;`
- Leitura: `int i = twoDim[0][0];`



## Modelo de Memória: Arrays Multidimensionais

### Da lista para a tabela

- Também chamado de **matriz**
- Composto por um **conjunto de arrays**

### Em Java

- Similar ao array
- `int twoDim[] [] = new int[4][5];`
- Escrita: `twoDim[0][0] = 8;`
- Leitura: `int i = twoDim[0][0];`

### Outras tipos derivados

- |                     |                 |
|---------------------|-----------------|
| ➤ <b>Vector</b>     | ➤ <b>Map</b>    |
| ➤ <b>ArrayList</b>  | ➤ <b>Object</b> |
| ➤ <b>String</b>     |                 |
| ➤ <b>Collection</b> | ➤ <b>Enums*</b> |



# Escopos de Variáveis

## Três tipos de escopos

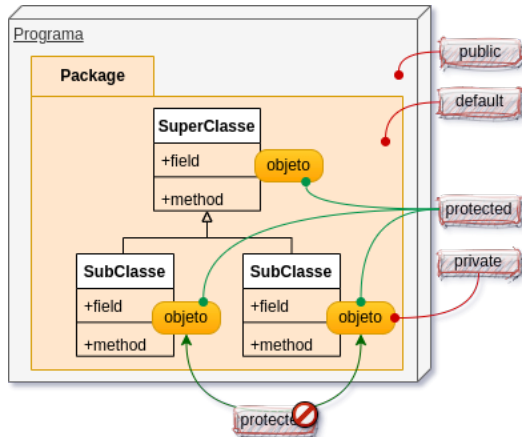
- **Variáveis locais** – Declaradas no **corpo de um método/função**. Ciclo de vida inicia e encerra quando o método inicia e finaliza, respectivamente.
- **Variáveis de instância** – **Visível** dentro de um objeto. É criada junto com o objeto
- **Variáveis estáticas** – Pertencem à classe. Criadas assim que o programa inicia a execução.

## Modificadores de acesso

- Os modificadores determinam a **“visibilidade”** de uma variável:
  - **public** – Acessível de **qualquer lugar**
  - **protected** – Acessível a partir da **classe** que pertence e suas **subclasses**
  - **default (package)** – Acessível no contexto do **namespace** da classe (**package**)
  - **private** – Acessível **apenas no objeto** da classe em que foi definida
- **Modificadores de acesso** também são usados para **métodos** e **classes**



## Resumindo Modificadores de Acesso





# Estruturas de Decisão

## Tipos de estruturas de decisão em Java

- Semelhante a outras Linguagens de programação
  - **if** – Estrutura mais simples de decisão
  - **if-else** – Adiciona à estrutura uma **ação alternativa** caso a **condição verificada** seja **falsa**
  - **nested-if** – Estruturas condicionais com **if/if-else** podem ser construídas dentro de outras estruturas
  - **if-else if** – É a formação mais **complexa** da estrutura de decisão. Permite que a decisão seja tomada com base em várias **condições**. Se a **condição verificada** for **verdadeira**, o conjunto de ações relacionado a ela será executado
  - **switch-case** –





# Sintaxe da Estrutura If

## Sintaxe

```
1 if (condition_1){  
2     statement_1_1;  
3     ...  
4 } else if (condition_2){  
5     statement_2_1;  
6     statement_2_2;  
7     ...  
8 } else {  
9     statement_else_1;  
10    ...  
11 }
```

## Exemplo

```
1 if (nota >= 7.0 && nota <= 10.0){  
2     ...  
3 } else if (nota >= 0 && nota < 5.0){  
4     ...  
5 } else if (nota >= 5.0 && nota < 7.0){  
6     ...  
7 } else {  
8     ...  
9 }
```







## Sintaxe da Estrutura Switch-Case

### O que é

- Instrução de **ramificação multidirecional**
- É uma maneira simples de **despachar** a execução para diferentes partes do código
- Muito usado para a construção de *menus* de opções de **sistemas de interface textual**
- Tem o mesmo efeito que a estrutura **if-else-if**

```
tercio@tercio-ZenBook-UX434FAC: ~/aulas-preparacao/codes/python
Arquivo  Editar  Ver  Pesquisar  Terminal  Ajuda
tercio@tercio-ZenBook-UX434FAC:~/aulas-preparacao/codes/python$ python3 Agenda.py
###   AGENDA DO Joaozinho de Deus
Opções
    1 - Preencher nome
    2 - Preencher endereço
    3 - Adicionar um novo contato
    4 - Listar contatos
    5 - Sair
Escolha uma opção |
```



# Sintaxe da Estrutura Switch-Case



## Sintaxe

```
1 switch (expression){  
2     case value1:  
3         statement1;  
4         break;  
5     case value2:  
6         statement2;  
7         break;  
8     ...  
9     case valueN:  
10        statementN;  
11        break;  
12    default:  
13        statementDefault;  
14 }
```

```
1 import java.io.*;  
2 public class SwitchDemo {  
3     public static void main (String[] args) {  
4         int num=20;  
5         switch (num){  
6             case 5:  sysout("It is 5");  
7                     break;  
8             case 10 : sysout("It is 10");  
9                     break;  
10            case 15 : sysout("It is 15");  
11                    break;  
12            case 20 : sysout("It is 20");  
13                    break;  
14            default: sysout("Not present");  
15        }  
16    }  
17 }
```



## Jump – Transferência de Controle

### Tipos de jumps

- Comandos que transferem o controle para outra parte do programa
- Java suporta três tipos de *jump*:
  - **Break** – utilizado principalmente
    - Encerrar uma sequência de instrução **switch-case**
    - Sair de um **laço** (*loop*)
  - **Continue** – encerra a **iteração corrente** de um laço de repetição
  - **Return** – usado para **retornar** um valor resultante de uma função e **transferir o controle de execução** para quem a chamou



# Operadores Lógicos

- **E** – &&
- **OU** – ||
- **NEGAÇÃO** – !



# Estruturas de Repetição

## Três tipos de estrutura de repetição

- **while** – A condição para continuar a repetição é **verificada no início** da iteração
- **do-while** – A condição para continuar a repetição é **verificada no final** da iteração
- **for** – O número de iterações é **predefinido** antes da estrutura ser executada



# Estruturas de Repetição

## Três tipos de estrutura de repetição

- 🔧 **while** – A condição para continuar a repetição é **verificada no início** da iteração
- 🔧 **do-while** – A condição para continuar a repetição é **verificada no final** da iteração
- 🔧 **for** – O número de iterações é **predefinido** antes da estrutura ser executada

```
1 while (condition){  
2     [loop statement]  
3 }
```

```
1 import java.io.*;  
2 public class WhileDemo {  
3     public static void main (String[] args) {  
4         int i = 0;  
5         while (i <= 10) {  
6             System.out.println(i);  
7             i++;  
8         }  
9     }  
}
```





# Estruturas de Repetição

## Três tipos de estrutura de repetição

- 🔧 **while** – A condição para continuar a repetição é **verificada no início** da iteração
- 🔧 **do-while** – A condição para continuar a repetição é **verificada no final** da iteração
- 🔧 **for** – O número de iterações é **predefinido** antes da estrutura ser executada

```
1 for (init; test; step) {  
2     [loop statement]  
3 }
```

```
1 import java.io.*;  
2 public class ForDemo {  
3     public static void main (String[] args){  
4         for (int i = 0; i <= 10; i++) {  
5             System.out.println(i);  
6         }  
7     }  
8 }
```





# Estruturas de Repetição

## Três tipos de estrutura de repetição

- 🔧 **while** – A condição para continuar a repetição é **verificada no início** da iteração
- 🔧 **do-while** – A condição para continuar a repetição é **verificada no final** da iteração
- 🔧 **for** – O número de iterações é **predefinido** antes da estrutura ser executada

```
1 do {  
2     [loop statement]  
3 } while (condition);
```

```
1 import java.io.*;  
2 public class DoWhileDemo {  
3     public static void main (String[] args) {  
4         int i = 0;  
5         do {  
6             System.out.println(i);  
7             i++;  
8         } while (i > 0 && i <= 10);  
9     }
```







# Funções (Métodos)

## Características das funções em Java

- Em OO, **função** é chamada de **método**
- São dois tipos de métodos em Java:
  - **Métodos estáticos** – **não dependem** da criação de um objeto para serem chamados
  - **Métodos de instância** – pertencem ao objeto, ou seja, **o objeto precisa existir** para poder chamar o método
- Método de instância será explicado melhor em Orientação a Objetos

```
1 public static <ReturnType> <methodName>(<parameters>){  
2     <statements>;  
3     return <value>;  
4 }
```



# Entrada de Dados pelo Usuário

## Como funciona

- Java recebe dados externos através do **pacote I/O**
- Os dados são transferidos através de **fluxos** (*streams*)

## Scanner

- Últimas versões de Java
- Formata por tipo de dado (*cast*):
  - Integer: **nextInt()**
  - Float: **nextFloat()**
  - String : **next()** and **nextLine()**

```
1 import java.util.Scanner;  
2 public class ScannerDemo{  
3     public static void main(String[] args) {  
4         Scanner scn = new Scanner(System.in);  
5         String entrada = scn.next();  
6     }  
7 }
```





## Mão na Massa

- A velha e boa **Agenda**
- Dados da agenda de contatos:
  - **Nome**
  - **Telefone**
  - **Endereço**
  - **Curso**
  - **Nível**
- Funções – **CRUD**:
  - **Criar**
  - **Ler** (buscar por nome e mostrar dados)
  - **Atualizar** (buscar e editar dados)
  - **Apagar** (buscar excluir)





# Interface Gráfica com JavaFX

## JavaFX

- **Plataforma** de criação de **aplicações gráficas** usando Java
- **JavaFX** substitui **AWT (*Abstract Window Toolkit*)** e **Swing**

## Características

- **Suporte a UI Moderna**
  - Interfaces **dinâmicas** e **modernas**
  - Suporte a efeitos visuais **2D** e **3D**
- **Simples e intuitiva**: projetada para ser simples e fácil de usar
- **CSS para estilos**: similar ao desenvolvimento web
- **Compatibilidade Multiplataforma**: windows, MacOS e Linux



# Estrutura Básica de uma Aplicação JavaFX

## Stage (palco)

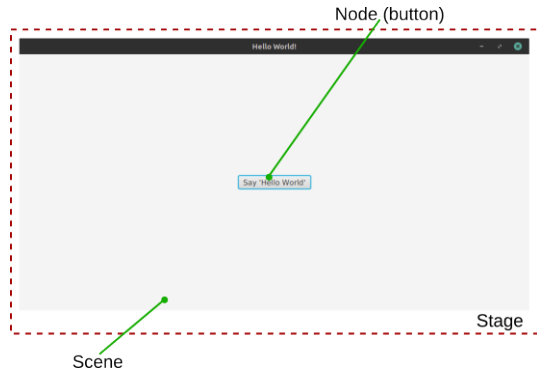
➡ Janela principal da aplicação

## Scene (sena)

➡ Contém a estrutura da interface gráfica, incluindo *layouts* e componentes

## Nodes

➡ Os elementos gráficos que compõem a interface, como **botões**, **textos**, **imagens**, etc





## Preparando o Ambiente

- **JavaFX** é uma **biblioteca modular independente**
  - Precisa ser baixada em <https://gluonhq.com/products/javafx/>
  - A distribuição deve ser **compatível** com o **hardware** e **sistema operacional**
- Cada projeto Desktop Java precisa dos arquivos **.jar** em sua pasta **lib/** ou uma referência ao local onde se encontram os arquivos **.jar**

### Passo a passo

- Seguiremos o passo a passo para a IDE **Visual Studio Code**
  - <https://openjfx.io/openjfx-docs/#introduction>



# Praticando

## “Desenhando” a Primeira tela

### 1º passo

- ➡ Crie um projeto Java no VSCode
- ➡ Adicione os arquivos **.jar** à biblioteca do projeto

### 2º Passo

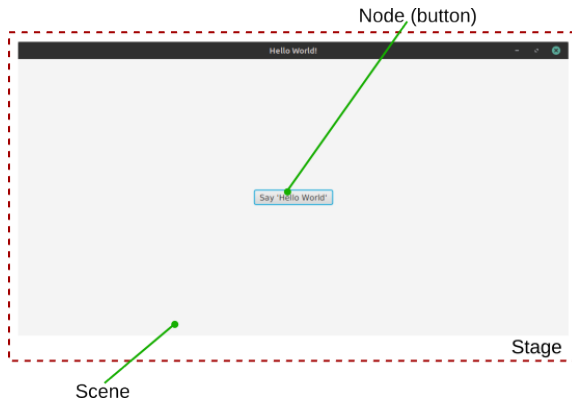
- ➡ Crie uma classe **SimpleJavaFXApp**
  - ➡ **Subclasse** de **Application**
  - ➡ **Importar** classes de **javaFX**

```
1 import javafx.application.  
   Application;  
2 import javafx.scene.Scene;  
3 import javafx.scene.control.Button;  
4 import javafx.scene.control.Label;  
5 import javafx.scene.control.  
   TextField;  
6 import javafx.scene.layout.VBox;  
7 import javafx.stage.Stage;  
8  
9 public class SimpleJavaFXApp  
10     extends Application {  
11     ...  
12 }
```



# Praticando

## Relembrando a Estrutura da Tela








# Praticando

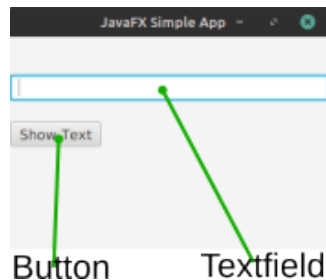
## Construindo Nodes

 **Construindo** os componentes da tela

```
1 TextField textField = new TextField();  
2 Button button = new Button("Mostrar Texto");  
3 Label lbl = new Label();
```

 Definindo a **ação** do botão **btn**

```
1 button.setOnAction(event -> {  
2     String text = textField.getText();  
3     lbl.setText(text);  
4 });
```



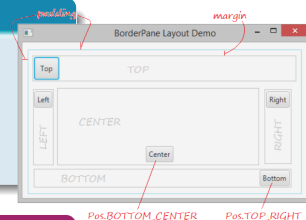
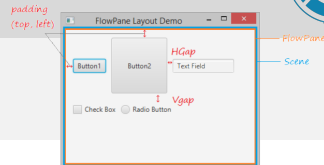


# Praticando

## Organizando os Componentes na Tela – *Layout*

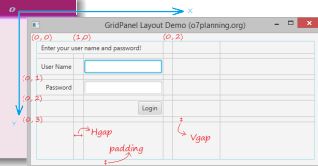
### Layouts da tela

- 🛠 **Nodes** do tipo **Containers** que gerenciam a disposição de **nodes** na tela
- 🛠 **Nodes** devem ser agrupados em **layouts** – **Painéis**
- 🛠 Um painel **pode conter** outro painel



### Tipos de painéis

- |                           |                       |
|---------------------------|-----------------------|
| 🛠 HBox, VBox e ButtonPane | 🛠 FlowPane e TilePane |
| 🛠 StackPane               | 🛠 BorderLayout        |
| 🛠 AnchorPane              | 🛠 SplitPane           |
| 🛠 GridPane                |                       |





# Praticando

Definindo um *layout* e adicionando os componentes

➡ Criando o *layout* **VBox** e adicionando componentes **nodes**

```
1 VBox vbox = new VBox(textField, button, lbl); // Criando o layout
2 vbox.setPadding(new Insets(10,10,10,10)); // Margens do painel
3 vbox.setSpacing(10); // Espaçamento entre nodes
```

➡ Criando uma *cena* e definindo-a no “palco” - **stage**

```
1 Scene scene = new Scene(vbox, 300, 200); // Criação da cena
2 primaryStage.setScene(scene); // Definição no palco
3 primaryStage.show(); // Apresentação da cena
```



# Praticando

Definindo um *layout* e adicionando os componentes

➡ Criando o *layout* **VBox** e adicionando componentes **nodes**

```
1 VBox vbox = new VBox(textField, button, lbl); // Criando o layout
2 vbox.setPadding(new Insets(10,10,10,10)); // Margens do painel
3 vbox.setSpacing(10); // Espaçamento entre nodes
```

➡ Criando uma **cena** e definindo-a no “palco” - **stage**

```
1 Scene scene = new Scene(vbox, 300, 200); // Criação da cena
2 primaryStage.setScene(scene); // Definição no palco
3 primaryStage.show(); // Apresentação da cena
```

➡ *Juntando tudo ...*



# Praticando

Voltando para a Classe ...

Lembram no início – classe **SimpleJavaFXApp?**

```
1 public class HelloWorldFX extends Application {  
2  
3     @Override  
4     public void start(Stage primaryStage) {  
5         ... // Nosso código vai aqui  
6             // é aqui que tudo acontece!  
7     }  
8  
9     public static void main(String[] args) {  
10         launch(args);  
11     }  
12 }
```



# Praticando

## Configurando a execução ...

2. Clique em Executar/Run  
3. Clique em Adicionar configuração...

1. Existe o arquivo?

```
1 {  
2   // Use o IntelliSense para saber mais sobre os atributos possíveis.  
3   // Focalizar para exibir as descrições dos atributos existentes.  
4   // Para obter mais informações, acesse: https://go.microsoft.com/fwlink/?linkid=830387  
5   "version": "0.2.0",  
6   "configurations": [  
7     {  
8       "type": "java",  
9       "name": "Current File",  
10      "request": "launch",  
11      "mainClass": "${file}"  
12    },  
13    {  
14      "type": "java",  
15      "name": "SimpleJavaFXApp",  
16      "request": "launch",  
17      "mainClass": "SimpleJavaFXApp",  
18      "projectName": "helloJavaFX 6a4e84aa",  
19      "vmArgs": "--module-path /media/tercio/docs/javafx-sdk-22.0.1/lib --add-modules javafx.controls,javafx.fxml"  
20    }  
21  ]  
22 }
```

4. Adicione a linha de argumentos  
module-path é o local onde estão os arquivos .jar

Adicionar Configuração...



# Implementação de uma Calculadora



- Uma aplicação simples de calculadora
- Entrada numérica em um campo de dados **TextField**
- Implementação de **botões numéricos** e **operações básicas** (adição, subtração, multiplicação, divisão)
- Exibição do resultado em texto **Label**
- Botão **“Limpar”** que limpa o campo numérico (**TextField**) e o texto (**Label**) do resultado da calculadora.