



Interface Gráfica com JavaFX

Prof. Evandro César Freiburger

Instituto Federal de Mato Grosso
Departamento da Área de Informática
evandro.freiberger@cba.ifmt.edu.br

2018

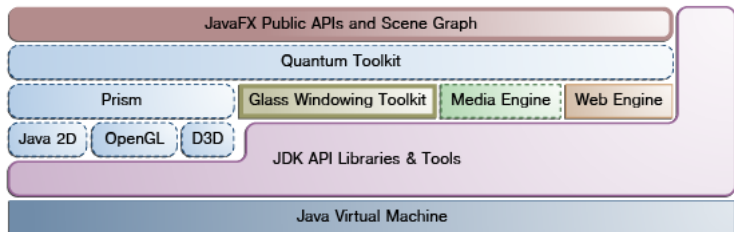
Sumário

- 1 Introdução
- 2 Exemplo Básico de Aplicação JavaFX
- 3 Gerenciadores de Layout
- 4 Controles do JavaFX
- 5 Tratamento de Eventos no JavaFX
- 6 Uso de CSS no JavaFX
- 7 Construindo Interface JavaFX com FXML

Introdução ao JavaFX

- O JavaFX é um conjunto de bibliotecas Java para produção de interfaces gráficas;
- Projetado para permitir que os desenvolvedores possam implementar interfaces ricas *Rich Client Platform (RCP)*;
- Mantém o comportamento consistente em múltiplas plataformas.

Arquitetura JavaFX



- Prism - Motor Gráfico do JavaFX
- Glass Windowing Toolkit - Sistema de Janelas
- Media Engine - Motor de Mídias
- Web Engine - Motor Web

Elementos da Arquitetura JavaFX

- **JavaFX Public APIs** - Interface de programação pública do JavaFX. Consiste dos elementos visíveis aos desenvolvedores de aplicativos JavaFX.
- **Scene Graph** - o Grafo de Cena é o ponto de partida para a construção de um aplicativo JavaFX. É uma árvore hierárquica de nós que representam todos os elementos visuais de uma interface com o usuário.
 - ▶ Um único elemento em um grafo de cena é chamado de nó;
 - ▶ Cada nó tem um ID (identificação) classe de estilo e um volume delimitado;
 - ▶ Com a exceção do nó raiz de um grafo de cena, cada nó tem um único pai e zero ou mais filhos;
 - ▶ Podem ter: efeitos de foco e sombra, opacidade, transformações geométricas e manipulação de eventos;

Estratégia de Desenvolvimento de Aplicações JavaFX

Existem duas alternativas de desenvolvimento de aplicações JavaFX, tais como:

- Codificação dos elementos da interface em código Java:
 - ▶ (+) Evita que o desenvolvedor tenha que conhecer outra linguagem para a definição dos elementos que compõem a interface gráfica.
 - ▶ (+) Evita a interpretação, em tempo de execução, do arquivo XML;
 - ▶ (-) Codificação extensa e complexa para a definição do layout da interface;
- Codificação dos elementos da interface com marcação XML:
 - ▶ (+-) O desenvolvedor tem que conhecer outra linguagem para a definição dos elementos que compõem a interface gráfica;
 - ▶ (-) Exige a interpretação, em tempo de execução, do arquivo XML;
 - ▶ (+) É possível produzir layout de interface por meio de ferramentas (SceneBuilder);
 - ▶ (+) Separação completa do layout e controlador;

Aplicação JavaFX

O primeiro exemplo será criado em um projeto do tipo Aplicação Java (Java Application).

Nessa forma não será usado recursos da IDE para a produção de código e a codificação dos elementos da interface será realizada por meio de código nativo JavaFX.

Passo 01: Crie um projeto do tipo Aplicação Java (Java Application)

- Salve o projeto com o nome *ProjetoJavaFXAplicacao01*;
- Crie um pacote com o nome de *visao*;
- Crie uma classe Java com o nome *Aplicacao01* no pacote *visao*;

Aplicação JavaFX

Passo 02: Codifique a classe *Aplicacao01* conforme código a seguir.

```
1 package visao ;
2
3 import javafx.application.Application ;
4 import javafx.scene.Scene ;
5 import javafx.scene.layout.AnchorPane ;
6 import javafx.stage.Stage ;
7
8 public class Aplicacao01 extends Application { //1
9
10     public static void main( String [] args ) { //2
11         Aplicacao01 .launch( args ) ;
12     }
13     @Override
14     public void start( Stage palco ) throws Exception { //3
15         AnchorPane painel = new AnchorPane() ; //4
16         Scene cena = new Scene( painel , 400 , 200 ) ; //5
17         palco.setScene( cena ) ; //6
18         palco.setTitle( "Titulo da Aplicacao01" ) ; //7
19         palco.show() ; //8
20     }
21 }
```


Aplicação JavaFX (1)

Pontos importantes de uma aplicação com JavaFX

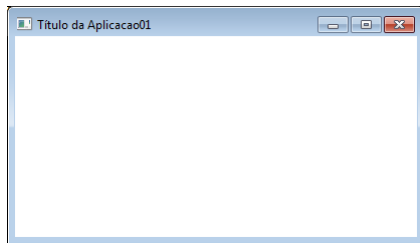
- 1 Toda classe principal de JavaFX deve herdar de *Application*;
- 2 No método *main* deve ser invocado o método *launch* para iniciar a execução da aplicação. No método *main* não vai código JavaFX, o código vai no método *start*;
- 3 O método *start*, herdado da classe *Application*, deve ser sobrescrito. O parâmetro recebido é do tipo *Stage* (palco), que pode ser interpretado como a estrutura da janela que será construída;
- 4 O elemento **painel**, do tipo *AnchorPane*, funciona como um *container* para adicionar os controles (elementos) que compõem a interface;

Aplicação JavaFX (2)

- 5 O elemento **cena**, do tipo *Scene*, representa o *container* principal de todos os elementos da aplicação JavaFX. No construtor do objeto cena é informado o elemento principal da cena e as dimensões (altura e largura) da cena;
- 6 O palco precisa de uma cena, isso é feito por meio do método *setScene()*, cujo parâmetro é o objeto **cena** já definido;
- 7 Na sequência é definido o título da janela que será exibida, por meio do método *setTitle()* do objeto **palco**;
- 8 Por último, deve ser invocado o método *show()* do objeto **palco**, para que o palco e cena sejam exibidos.

Aplicação JavaFX

Tela da Aplicacao01 exibida com a execução da aplicação.



O painel é exibido em branco, pois não foram inseridos outros componentes.

Aplicação JavaFX (1)

Codificação da Classe Aplicacao02 com a inclusão de um controle (Label) no painel.

```
1 package visao;
2 import javafx.application.Application;
3 import javafx.scene.Scene;
4 import javafx.scene.control.Label;
5 import javafx.scene.layout.AnchorPane;
6 import javafx.scene.text.Font;
7 import javafx.scene.text.FontWeight;
8 import javafx.stage.Stage;
9 public class Aplicacao02 extends Application {
10
11     @Override
12     public void start(Stage palco) throws Exception {
13         AnchorPane painel = new AnchorPane();
14
15         Label label1 = new Label();
16         label1.setText("Texto do Label");
17         label1.setFont(Font.font("Arial", FontWeight.BOLD, 14));
18         AnchorPane.setTopAnchor(label1, 20.0);
19         AnchorPane.setLeftAnchor(label1, 20.0);
20         painel.getChildren().add(label1);
21
22         Scene cena = new Scene(painel, 400, 200);
23         palco.setTitle("Aplicacao02");
24         palco.setScene(cena);
25         palco.show();
26     }
```

Aplicação JavaFX (2)

```
27  
28     public static void main(String[] args) {  
29         Aplicacao02.launch(args);  
30     }  
31 }
```

Aplicação JavaFX

Tela da Aplicacao02.



O painel é exibido com um componente Label.

Gerenciadores de Layout

Em aplicações com interface gráfica o ideal é permitir que o usuário redimensione uma área visível e todos os controles de UI também sejam redimensionados, a fim de proporcionar uma experiência de usuário agradável.

Semelhante a API Swing de Java, a API do JavaFX possui gerenciadores layouts que proporcionam as formas mais comuns para mostrar os controles de interface do usuário para o grafo de cena. Os layouts JavaFX são:

- `javafx.scene.layout.HBox`
- `javafx.scene.layout.VBox`
- `javafx.scene.layout.FlowPane`
- `javafx.scene.layout.BorderPane`
- `javafx.scene.layout.GridPane`

Layout HBox

O gerenciador HBox coloca os nós filhos JavaFX em uma linha horizontal. Como nós filho são adicionados consecutivamente, cada um é acrescentado ao final (lado direito).

Por padrão, o layout HBox considera a largura e altura preferida do nó filho.

Quando o nó pai não é redimensionável (um nó de grupo, por exemplo), a altura da linha do HBox assume a altura do nó filho com a maior altura preferida.

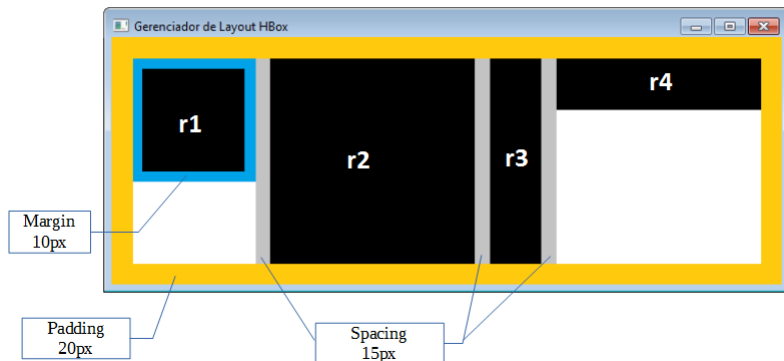
Além disso, por padrão, cada nó filho se alinha com a posição superior esquerdo (Pos.TOP-LEFT).

Layout HBox

```
1 package visao;
2 import javafx.application.Application;
3 import javafx.geometry.Insets;
4 import javafx.scene.Scene;
5 import javafx.scene.layout.HBox;
6 import javafx.scene.shape.Rectangle;
7 import javafx.stage.Stage;
8 public class LayoutHBox extends Application {
9
10     public static void main(String[] args) {
11         LayoutHBox.launch(args);
12     }
13
14     @Override
15     public void start(Stage palco) throws Exception {
16
17         HBox hbox = new HBox(15); // espaco entre os nos filhos
18         hbox.setPadding(new Insets(20)); // preenchimento entre a borda hbox e a linha.
19         Rectangle r1 = new Rectangle(100, 100); // quadrado 1
20         Rectangle r2 = new Rectangle(200, 200); // quadrado 2
21         Rectangle r3 = new Rectangle(50, 200); // retangulo vertical
22         Rectangle r4 = new Rectangle(200, 50); // retangulo horizontal
23         HBox.setMargin(r1, new Insets(10, 10, 10, 10)); // margem de um no filho especifico
24         hbox.getChildren().addAll(r1, r2, r3, r4);
25
26         Scene cena = new Scene(hbox);
27         palco.setScene(cena);
28         palco.setTitle("Gerenciador de Layout HBox");
29         palco.show();
30     }
31 }
```

Layout HBox

Ilustração da distribuição dos elementos pelo HBox



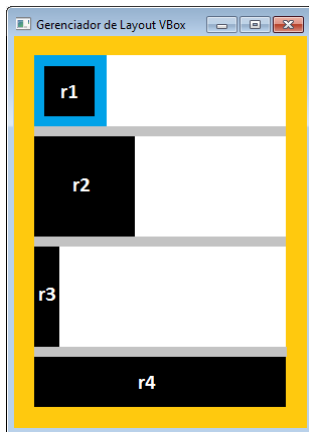
Layout VBox

O gerenciador VBox é similar ao HBox, exceto pelo fato que coloca os nós filhos JavaFX em uma coluna vertical.

```
1 package visao;
2 import javafx.application.Application;
3 import javafx.geometry.Insets;
4 import javafx.scene.Scene;
5 import javafx.scene.layout.VBox;
6 import javafx.scene.shape.Rectangle;
7 import javafx.stage.Stage;
8 public class LayoutVBox extends Application {
9     public static void main(String[] args) {
10         LayoutVBox.launch(args);
11     }
12     @Override
13     public void start(Stage palco) throws Exception {
14
15         VBox vbox = new VBox(10); // espaco entre os nos filhos
16         vbox.setPadding(new Insets(20)); // preenchimento entre a borda vbox e a linha.
17         Rectangle r1 = new Rectangle(50, 50); // quadrado 1
18         Rectangle r2 = new Rectangle(100, 100); // quadrado 2
19         Rectangle r3 = new Rectangle(25, 100); // retangulo vertical
20         Rectangle r4 = new Rectangle(250, 50); // retangulo horizontal
21         VBox.setMargin(r1, new Insets(10, 10, 10, 10)); // margem de um no filho especifico
22         vbox.getChildren().addAll(r1, r2, r3, r4);
23
24         Scene cena = new Scene(vbox);
25         palco.setScene(cena);
26         palco.setTitle("Gerenciador de Layout VBox");
27         palco.show();
28     }
29 }
```

Layout VBox

Ilustração da distribuição dos elementos pelo VBox



Layout FlowPane (1)

O gerenciador FlowPane distribui os componentes de interface em linhas/colunas, seguindo o fluxo de inserção dos componentes no componente pai.

Quando o componente pai é redimensionado os componentes filhos são reorganizados.

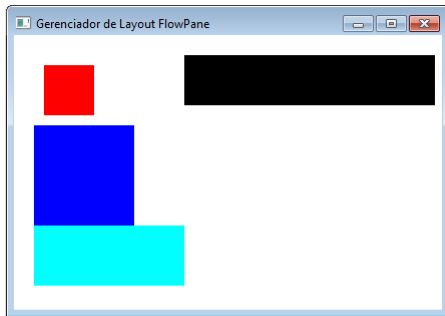
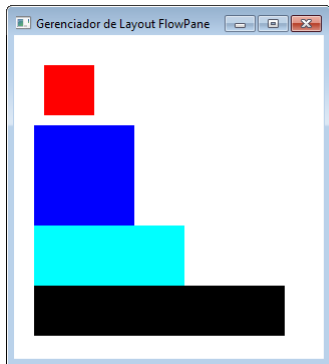
```
1 package visao;  
2 import javafx.application.Application;  
3 import javafx.geometry.Insets;  
4 import javafx.geometry.Orientation;  
5 import javafx.geometry.Pos;  
6 import javafx.scene.Scene;  
7 import javafx.scene.layout.FlowPane;  
8 import javafx.scene.paint.Color;  
9 import javafx.scene.shape.Rectangle;  
10 import javafx.stage.Stage;  
11 public class LayoutFlowPane extends Application {  
12  
13     public static void main(String[] args) {  
14         LayoutFlowPane.launch(args);  
15     }  
16  
17     @Override  
18     public void start(Stage palco) throws Exception {  
19
```

Layout FlowPane (2)

```
20 FlowPane flowPane = new FlowPane(Orientation.VERTICAL); // fluxo de distribuicao
21 flowPane.setAlignment(Pos.TOP_LEFT);
22 flowPane.setPadding(new Insets(20)); // borda entre FlowPane e a linha/coluna.
23 Rectangle r1 = new Rectangle(50, 50); // quadrado 1
24 r1.setFill(Color.RED);
25 Rectangle r2 = new Rectangle(100, 100); // quadrado 2
26 r2.setFill(Color.BLUE);
27 Rectangle r3 = new Rectangle(150, 60); // retangulo vertical
28 r3.setFill(Color.AQUA);
29 Rectangle r4 = new Rectangle(250, 50); // retangulo horizontal
30 FlowPane.setMargin(r1, new Insets(10, 10, 10, 10)); // margem de um no filho especifico
31 flowPane.getChildren().addAll(r1, r2, r3, r4);
32
33 Scene cena = new Scene(flowPane);
34 palco.setScene(cena);
35 palco.setTitle("Gerenciador de Layout FlowPane");
36 palco.show();
37 }
38 }
```

Layout FlowPane

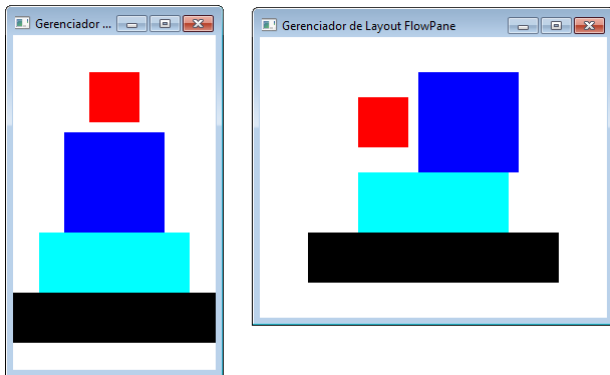
Ilustração da distribuição dos elementos pelo FlowPane.



```
FlowPane flowPane = new FlowPane(Orientation.VERTICAL);  
flowPane.setAlignment(Pos.TOP_LEFT);
```

Layout FlowPane

Ilustração da distribuição dos elementos pelo FlowPane.



```
FlowPane flowPane = new FlowPane(Orientation.HORIZONTAL);  
flowPane.setAlignment(Pos.CENTER);
```


Layout StackPane (1)

O gerenciador StackPane distribui os componentes de interface como uma pilha. Organiza os elementos de interface um sobre o outro conforme a ordem de inserção no painel pai.

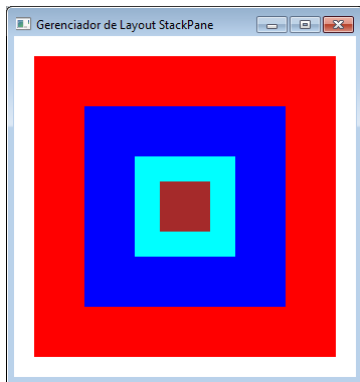
```
1 package visao;
2 import javafx.application.Application;
3 import javafx.geometry.Insets;
4 import javafx.scene.Scene;
5 import javafx.scene.layout.StackPane;
6 import javafx.scene.paint.Color;
7 import javafx.scene.shape.Rectangle;
8 import javafx.stage.Stage;
9 public class LayoutStackPane extends Application {
10
11     public static void main(String[] args) {
12         LayoutStackPane.launch(args);
13     }
14
15     @Override
16     public void start(Stage palco) throws Exception {
17
18         StackPane stackPane = new StackPane();
19         stackPane.setPadding(new Insets(20));
20         Rectangle r1 = new Rectangle(300, 300);
21         r1.setFill(Color.RED);
22         Rectangle r2 = new Rectangle(200, 200);
23         r2.setFill(Color.BLUE);
24         Rectangle r3 = new Rectangle(100, 100);
```

Layout StackPane (2)

```
25     r3.setFill(Color.AQUA);
26     Rectangle r4 = new Rectangle(50, 50);
27     r4.setFill(Color.BROWN);
28     stackPane.getChildren().addAll(r1, r2, r3, r4);
29
30     Scene cena = new Scene(stackPane);
31     palco.setScene(cena);
32     palco.setTitle("Gerenciador de Layout StackPane");
33     palco.show();
34 }
35 }
```

Layout StackPane

Ilustração da distribuição dos elementos pelo StackPane.



Layout AnchorPane (1)

O gerenciador AnchorPane distribui os componentes de interface ancorados nas extremidades TOP, LEFT, RIGHT E BOTTOM. Cada elemento pode estar ancorado em mais de uma das margens.

Caso o elemento esteja ancorado em lados opostos e este possa ser redimensionado, terá seu tamanho alterado quando um dos lados for reposicionado.

```
1 package visao;
2 import javafx.application.Application;
3 import javafx.scene.Scene;
4 import javafx.scene.layout.AnchorPane;
5 import javafx.scene.paint.Color;
6 import javafx.scene.shape.Rectangle;
7 import javafx.stage.Stage;
8 public class LayoutAnchorPane extends Application {
9
10     public static void main(String[] args) {
11         LayoutAnchorPane.launch(args);
12     }
13
14     @Override
15     public void start(Stage palco) throws Exception {
16
17         AnchorPane anchorPane = new AnchorPane();
```

Layout AnchorPane (2)

```
18 Rectangle r1 = new Rectangle(150, 60); // quadrado 1
19 r1.setFill(Color.RED);
20 Rectangle r2 = new Rectangle(150, 60); // quadrado 2
21 r2.setFill(Color.BLUE);
22 Rectangle r3 = new Rectangle(150, 60); // retangulo vertical
23 r3.setFill(Color.AQUA);
24 Rectangle r4 = new Rectangle(150, 60); // retangulo horizontal
25 r4.setFill(Color.BROWN);
26 Rectangle r5 = new Rectangle(150, 60); // retangulo horizontal
27
28 AnchorPane.setTopAnchor(r1, 20.0);
29 AnchorPane.setTopAnchor(r2, 40.0);
30 AnchorPane.setLeftAnchor(r2, 160.0);
31 AnchorPane.setRightAnchor(r3, 30.0);
32 AnchorPane.setTopAnchor(r3, 20.0);
33 AnchorPane.setBottomAnchor(r4, 20.0);
34 AnchorPane.setTopAnchor(r5, 100.0);
35
36 anchorPane.getChildren().addAll(r1, r2, r3, r4, r5);
37 Scene cena = new Scene(anchorPane, 500, 300);
38 palco.setScene(cena);
39 palco.setTitle("Gerenciador de Layout AnchorPane");
40 palco.show();
41 }
42 }
```

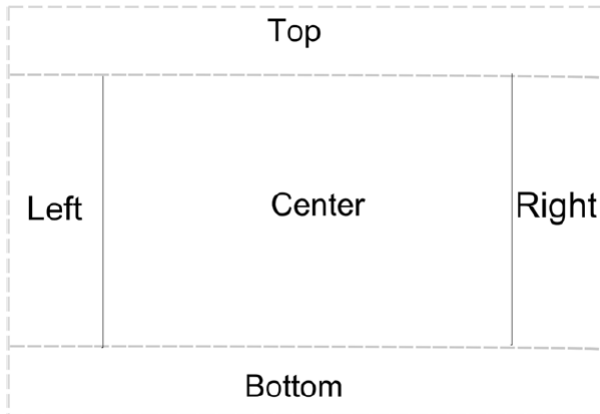
Layout AnchorPane

Ilustração da distribuição dos elementos pelo AnchorPane.



Layout BorderPane

O layout BorderPane permite a distribuição dos elementos da interface em cinco regiões.



Layout BorderPane (1)

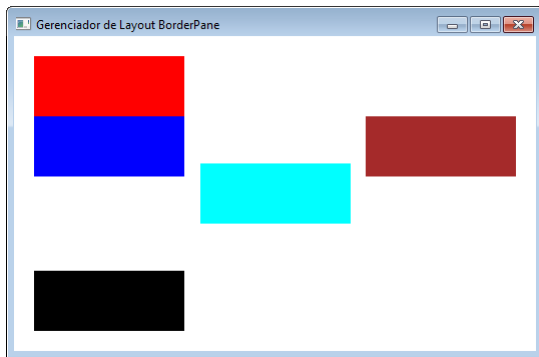
```
1 package visao;
2 import javafx.application.Application;
3 import javafx.geometry.Insets;
4 import javafx.geometry.Pos;
5 import javafx.scene.Scene;
6 import javafx.scene.layout.BorderPane;
7 import javafx.scene.paint.Color;
8 import javafx.scene.shape.Rectangle;
9 import javafx.stage.Stage;
10 public class LayoutBorderPane extends Application {
11
12     public static void main(String[] args) {
13         LayoutBorderPane.launch(args);
14     }
15
16     @Override
17     public void start(Stage palco) throws Exception {
18
19         BorderPane borderPane = new BorderPane();
20         borderPane.setPadding(new Insets(20)); // borda entre FlowPane e a linha/coluna.
21         Rectangle r1 = new Rectangle(150, 60); // quadrado 1
22         r1.setFill(Color.RED);
23         Rectangle r2 = new Rectangle(150, 60); // quadrado 2
24         r2.setFill(Color.BLUE);
25         Rectangle r3 = new Rectangle(150, 60); // retangulo vertical
26         r3.setFill(Color.AQUA);
27         Rectangle r4 = new Rectangle(150, 60); // retangulo horizontal
28         r4.setFill(Color.BROWN);
29         Rectangle r5 = new Rectangle(150, 60); // retangulo horizontal
30     }
```


Layout BorderPane (2)

```
31     BorderPane.setAlignment(r1, Pos.CENTER);
32     BorderPane.setAlignment(r2, Pos.CENTER);
33     BorderPane.setAlignment(r4, Pos.CENTER);
34     BorderPane.setAlignment(r5, Pos.CENTER);
35
36     borderPane.setTop(r1);
37     borderPane.setLeft(r2);
38     borderPane.setCenter(r3);
39     borderPane.setRight(r4);
40     borderPane.setBottom(r5);
41
42     Scene cena = new Scene(borderPane);
43     palco.setScene(cena);
44     palco.setTitle("Gerenciador de Layout BorderPane");
45     palco.show();
46 }
47 }
```

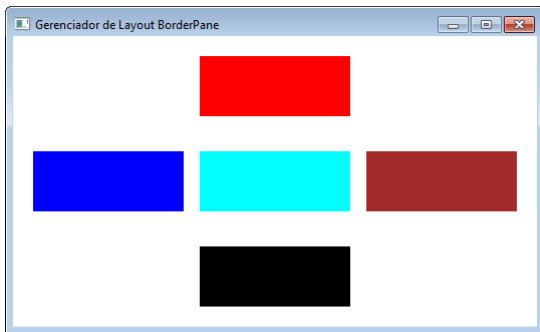
Layout BorderLayout

Ilustração da distribuição dos elementos pelo BorderLayout.



Layout BorderLayout

Ilustração da distribuição dos elementos pelo BorderLayout.



```
BorderPane.setAlignment(r1, Pos.CENTER);  
BorderPane.setAlignment(r2, Pos.CENTER);  
BorderPane.setAlignment(r4, Pos.CENTER);  
BorderPane.setAlignment(r5, Pos.CENTER);  
  
borderPane.setTop(r1);  
borderPane.setLeft(r2);  
borderPane.setCenter(r3);  
borderPane.setRight(r4);  
borderPane.setBottom(r5);
```

Layout TilePane (1)

O gerenciador TilePane distribui os componentes de interface em grade de "azulejos" de tamanhos fixos baseados no maior tamanho entre os componentes (tamanho preferido). A grade pode ser organizada no sentido Horizontal (padrão) ou Vertical.

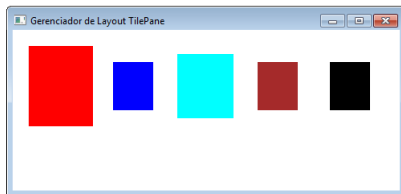
```
1 package visao ;
2 import javafx.application.Application ;
3 import javafx.geometry.Insets ;
4 import javafx.geometry.Orientation ;
5 import javafx.scene.Scene ;
6 import javafx.scene.layout.AnchorPane ;
7 import javafx.scene.layout.TilePane ;
8 import javafx.scene.paint.Color ;
9 import javafx.scene.shape.Rectangle ;
10 import javafx.stage.Stage ;
11 public class LayoutTilePane extends Application {
12
13     public static void main( String [] args ) {
14         LayoutTilePane.launch( args ) ;
15     }
16
17     @Override
18     public void start( Stage palco ) throws Exception {
19
20         TilePane tilePane = new TilePane( Orientation.HORIZONTAL ) ;
21         tilePane.setHgap( 10 ) ;
22         tilePane.setPrefColumns( 30 ) ;
```

Layout TilePane (2)

```
23 tilePane.setPadding(new Insets(20));
24 Rectangle r1 = new Rectangle(80, 100); // quadrado 1
25 r1.setFill(Color.RED);
26 Rectangle r2 = new Rectangle(50, 60); // quadrado 2
27 r2.setFill(Color.BLUE);
28 Rectangle r3 = new Rectangle(70, 80); // retangulo vertical
29 r3.setFill(Color.AQUA);
30 Rectangle r4 = new Rectangle(50, 60); // retangulo horizontal
31 r4.setFill(Color.BROWN);
32 Rectangle r5 = new Rectangle(50, 60); // retangulo horizontal
33
34 tilePane.getChildren().addAll(r1, r2, r3, r4, r5);
35 Scene cena = new Scene(tilePane, 500, 200);
36 palco.setScene(cena);
37 palco.setTitle("Gerenciador de Layout TilePane");
38 palco.show();
39 }
40 }
```

Layout TilePane

Ilustração da distribuição dos elementos pelo TilePane.



Layout GridPane (1)

O layout GridPane permite a distribuição dos elementos da interface em uma abstração de grade organizada em colunas e linhas.

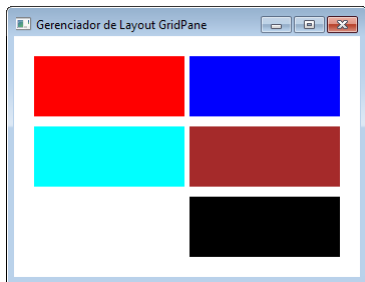
```
1 package visao;
2
3 import javafx.application.Application;
4 import javafx.geometry.Insets;
5 import javafx.scene.Scene;
6 import javafx.scene.layout.GridPane;
7 import javafx.scene.paint.Color;
8 import javafx.scene.shape.Rectangle;
9 import javafx.stage.Stage;
10
11 public class LayoutGridPane extends Application {
12
13     public static void main(String[] args) {
14         LayoutGridPane.launch(args);
15     }
16
17     @Override
18     public void start(Stage palco) throws Exception {
19
20         GridPane gridPane = new GridPane();
21         gridPane.setPadding(new Insets(20)); // borda entre GridPane e a linha/coluna.
22         gridPane.setHgap(5); //separacao horizontal entre as colunas
23         gridPane.setVgap(10); //separacao vertical entre as linhas
24
25         Rectangle r1 = new Rectangle(150, 60); // quadrado 1
```

Layout GridPane (2)

```
26 r1.setFill(Color.RED);
27 Rectangle r2 = new Rectangle(150, 60); // quadrado 2
28 r2.setFill(Color.BLUE);
29 Rectangle r3 = new Rectangle(150, 60); // retangulo vertical
30 r3.setFill(Color.AQUA);
31 Rectangle r4 = new Rectangle(150, 60); // retangulo horizontal
32 r4.setFill(Color.BROWN);
33 Rectangle r5 = new Rectangle(150, 60); // retangulo horizontal
34
35 gridPane.add(r1, 0, 0);
36 gridPane.add(r2, 1, 0);
37 gridPane.add(r3, 0, 1);
38 gridPane.add(r4, 1, 1);
39 gridPane.add(r5, 1, 2);
40
41 Scene cena = new Scene(gridPane);
42 palco.setScene(cena);
43 palco.setTitle("Gerenciador de Layout GridPane");
44 palco.show();
45 }
46 }
```


Layout GridPane

Ilustração da distribuição dos elementos pelo GridPane.



Controles do JavaFX

Uma das principais característica do JavaFX é oferecer muitos controles de interface, que vão de simples rótulos para mostrar texto até complexas tabelas.

Todos os controles do JavaFX herdam de forma direta ou indireta da classe `Control`.

```
javafx.scene.control
```

Class Control

```
java.lang.Object
  javafx.scene.Node
    javafx.scene.Parent
      javafx.scene.layout.Region
        javafx.scene.control.Control
```

All Implemented Interfaces:

`Styleable`, `EventTarget`, `Skinnable`

Direct Known Subclasses:

`Accordion`, `ButtonBar`, `ChoiceBox`, `ComboBoxBase`, `HTML editor`, `Labeled`, `ListView`, `MenuBar`, `Pagination`, `ProgressIndicator`, `ScrollBar`, `ScrollPane`, `Separator`, `Slider`, `Spinner`, `SplitPane`, `TableView`, `TabPane`, `TextInputControl`, `ToolBar`, `TreeTableView`, `TreeView`

Controles do JavaFX

Exemplos de Controles para construção de Interfaces Gráficas.



Controles do JavaFX (1)

Exemplos de utilização de Label, TextField, TextArea, Separator e Slider.

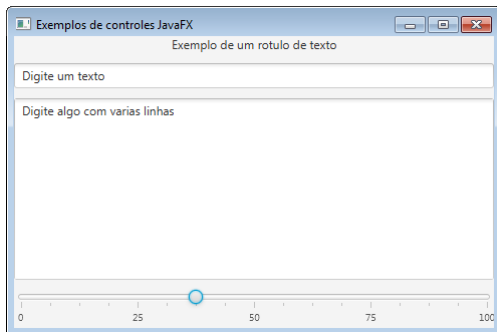
```
1 package visao ;
2
3 import javafx.application.Application ;
4 import javafx.geometry.Pos ;
5 import javafx.scene.Scene ;
6 import javafx.scene.control.Label ;
7 import javafx.scene.control.Slider ;
8 import javafx.scene.control.TextArea ;
9 import javafx.scene.control.TextField ;
10 import javafx.scene.control.Tooltip ;
11 import javafx.scene.layout.VBox ;
12 import javafx.stage.Stage ;
13
14 public class Controles01 extends Application {
15
16     public static void main(String[] args) {
17         Controles01.launch(args) ;
18     }
19
20     @Override
21     public void start(Stage palco) throws Exception {
22
23         VBox painelPrincipal = new VBox(10) ;
24         painelPrincipal.setAlignment(Pos.CENTER) ;
25         Label label1 = new Label("Exemplo de um rotulo de texto") ;
26         label1.setTooltip(new Tooltip("Label para mostrar textos simples")) ;
27         TextField campoTexto = new TextField("Digite um texto") ;
28         campoTexto.setTooltip(new Tooltip("Campo de texto para entrada de uma linha")) ;
```

Controles do JavaFX (2)

```
29  TextArea areaTexto = new TextArea("Digite algo com varias linhas");
30  areaTexto.setTooltip(new Tooltip("Campo de texto para entrada de multiplas linhas"));
31  Slider deslizando = new Slider();
32  deslizando.setShowTickLabels(true);
33  deslizando.setShowTickMarks(true);
34  deslizando.setTooltip(new Tooltip("Associa um valor numerico de acordo com sua posicao"));
35  painelPrincipal.getChildren().addAll(label1, campoTexto, areaTexto, deslizando);
36  Scene cena = new Scene(painelPrincipal);
37  palco.setScene(cena);
38  palco.setTitle("Exemplos de controles JavaFX");
39  palco.show();
40  }
41 }
```

Controles do JavaFX

Tela resultante da execução do exemplo anterior.



Controles do JavaFX (1)

Exemplo do uso do layout GridPane combinado com outros gerenciadores de layout e controles redimensionáveis.

```
1 package visao ;
2
3 import javafx.application.Application ;
4 import javafx.geometry.HPos ;
5 import javafx.geometry.Insets ;
6 import javafx.scene.Scene ;
7 import javafx.scene.control.Button ;
8 import javafx.scene.control.Label ;
9 import javafx.scene.control.TextField ;
10 import javafx.scene.layout.BorderPane ;
11 import javafx.scene.layout.ColumnConstraints ;
12 import javafx.scene.layout.FlowPane ;
13 import javafx.scene.layout.GridPane ;
14 import javafx.scene.layout.Priority ;
15 import javafx.scene.paint.Color ;
16 import javafx.scene.text.Font ;
17 import javafx.scene.text.Text ;
18 import javafx.stage.Stage ;
19
20 public class Controles02 extends Application {
21
22     public static void main( String [] args ) {
23         Controles02.launch( args ) ;
24     }
25 }
```

Controles do JavaFX (2)

```
26 @Override
27 public void start(Stage palco) throws Exception {
28
29     palco.setTitle("Formulario GridPane");
30     BorderPane panelPrincipal = new BorderPane();
31     Scene cena = new Scene(panelPrincipal, 380, 150, Color.WHITE);
32
33     GridPane gridPane = new GridPane();
34     gridPane.setPadding(new Insets(5));
35     gridPane.setHgap(5);
36     gridPane.setVgap(5);
37     ColumnConstraints column1 = new ColumnConstraints(100); //tamanho fixo
38     ColumnConstraints column2 = new ColumnConstraints(50, 150, 300); // min, preferido, max
39     column2.setHgrow(Priority.ALWAYS); //permite redimensionamento
40     gridPane.getColumnConstraints().addAll(column1, column2);
41
42     Label pNomeLabel = new Label("Primeiro Nome");
43     TextField pNomeCampo = new TextField();
44     Label uNomeLabel = new Label("Ultimo Nome");
45     TextField uNomeCampo = new TextField();
46     Button salvarBotao = new Button("Salvar");
47     GridPane.setHalignment(pNomeLabel, HPos.RIGHT);
48     gridPane.add(pNomeLabel, 0, 0);
49
50     GridPane.setHalignment(uNomeLabel, HPos.RIGHT);
51     gridPane.add(uNomeLabel, 0, 1);
52     GridPane.setHalignment(pNomeCampo, HPos.LEFT);
53     gridPane.add(pNomeCampo, 1, 0);
54     GridPane.setHalignment(uNomeCampo, HPos.LEFT);
55     gridPane.add(uNomeCampo, 1, 1);
56     GridPane.setHalignment(salvarBotao, HPos.RIGHT);
```


Controles do JavaFX (3)

```
57 gridPane.add(salvarBotao, 1, 2);
58
59 FlowPane cabecalhoTopo = new FlowPane();
60 cabecalhoTopo.setPrefHeight(40);
61 String backgroundStyle
62     = "-fx-background-color: lightblue;"
63       + "-fx-background-radius: 20%;"
64       + "-fx-background-inset: 10px;";
65 cabecalhoTopo.setStyle(backgroundStyle);
66
67 Text contatoTexto = new Text("Contatos");
68 contatoTexto.setFill(Color.WHITE);
69 Font serif = Font.font("Dialog", 30);
70 contatoTexto.setFont(serif);
71 cabecalhoTopo.getChildren().add(contatoTexto);
72 panelPrincipal.setTop(cabecalhoTopo);
73 panelPrincipal.setCenter(gridPane);
74 palco.setScene(cena);
75 palco.show();
76 }
77 }
```

Controles do JavaFX

Tela resultante da execução do exemplo anterior.



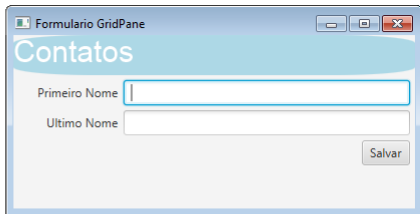
Formulario GridPane

Contatos

Primeiro Nome

Ultimo Nome

Salvar



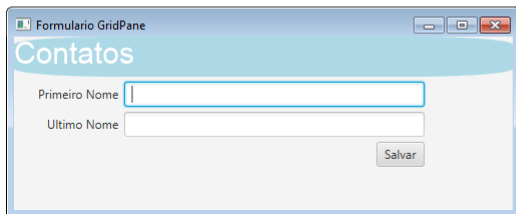
Formulario GridPane

Contatos

Primeiro Nome

Ultimo Nome

Salvar



Formulario GridPane

Contatos

Primeiro Nome

Ultimo Nome

Salvar

Tratamento de Eventos

Tratar evento é definir o comportamento da aplicação, por meio de código, que será executado de acordo com uma ação do usuário.

O usuário interage com a aplicação por meio dos controles da interface.

Por exemplo, passa o cursor do mouse sobre um componente, arrasta coisas, clica, move o cursor para dentro ou para fora de um componente, escolhe um elemento em uma listas, seleciona, entre outras.

O tratamento de eventos compreende duas etapas:

- 1 escrever o comportamento;
- 2 associar o código do comportamento ao controle e evento desejado.

Tratamento de Eventos

No JavaFX, o comportamento de um evento pode ser escrito de três formas distintas:

- 1 escrever uma classe específica para tratar o evento. A classe deve implementar uma Interface de tratamento de evento;
- 2 implementar uma classe interna e anônima. A partir da interface, define-se a implementação dos métodos da Interface de evento;
- 3 implementar na própria classe que define a interface gráfica. A classe que define a interface gráfica deve implementar a Interface de evento desejada;

Tratamento de Eventos

Tratamento do Evento OnAction do componente Button

- É chamado sempre que o botão é acionado;
- O evento é disparado quando o usuário clica no botão com o mouse, quando ocorre um toque em uma tela touch, quando uma tecla é pressionada ou quando o desenvolvedor programaticamente invoca o método `fire()` do botão;
- Para uma classe ser Ouvinte de um evento deve implementar a Interface relacionada ao evento, neste caso a interface `EventHandler`.

Tratamento de Eventos com Classe Externa

A classe externa *TratadorEventoOnAction* implementa o comportamento do evento.

O requisito é que realize a Interface *EventHandler*.

```
1 package visao;  
2  
3 import javafx.event.Event;  
4 import javafx.event.EventHandler;  
5  
6 public class TratadorEventoOnAction implements EventHandler{  
7  
8     @Override  
9     public void handle(Event event) {  
10         System.out.println("Evento tratado na classe externa "+  
11             "\nTipo Evento "+event.getEventType()+  
12             "\nOrigem "+event.getSource());  
13     }  
14 }
```

Tratamento de Eventos com Classe Externa (1)

A classe *TratamentoEventoBotao01* define a interface gráfica.

O comportamento do evento é implementado na classe externa *TratadorEventoOnAction* que realiza a Interface *EventHandler*.

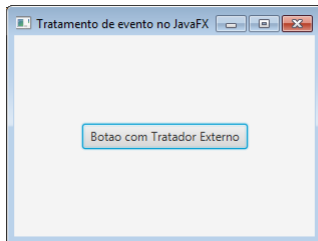
A Interface *EventHandler* permite que a classe *TratadorEventoOnAction* produza objetos que possam ser associados a componentes de interfaces que disparam eventos desse tipo.

```
1 package visao ;
2 import javafx.application.Application ;
3 import javafx.geometry.Pos ;
4 import javafx.scene.Scene ;
5 import javafx.scene.control.Button ;
6 import javafx.scene.layout.VBox ;
7 import javafx.stage.Stage ;
8 public class TratamentoEventoBotao01 extends Application {
9
10     public static void main(String [] args) {
11         TratamentoEventoBotao01 . launch ( args ) ;
12     }
13
14     @Override
15     public void start(Stage palco) throws Exception {
```

Tratamento de Eventos com Classe Externa (2)

```
16 VBox painelPrincipal = new VBox(10);
17 painelPrincipal.setAlignment(Pos.CENTER);
18 Button botao1 = new Button("Botao com Tratador Externo");
19
20 //cria uma instancia da classe externa para tratamento do evento
21 botao1.setOnAction(new TratadorEventoOnAction());
22
23 painelPrincipal.getChildren().add(botao1);
24 Scene cena = new Scene(painelPrincipal, 300, 200);
25 palco.setScene(cena);
26 palco.setTitle("Tratamento de evento no JavaFX");
27 palco.show();
28 }
29 }
```


Tratamento de Eventos com Classe Externa (3)



```
run:  
Evento tratado na classe externa  
Tipo Evento ACTION  
Origem Button@63e2a4d7[styleClass=button]'Botao com Tratador Externo'
```

Tratamento de Eventos com Classe Anônima (1)

A classe *TratamentoEventoBotao02* define a interface gráfica.

O comportamento do evento é implementado em uma classe anônima e interna que realiza a Interface *EventHandler*.

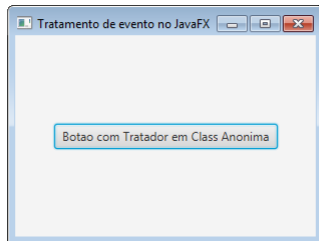
A Interface *EventHandler* permite que o objeto da classe anônima seja associado ao componente da interface que dispara o evento.

```
1 package visao ;
2 import javafx.application.Application ;
3 import javafx.event.Event ;
4 import javafx.event.EventHandler ;
5 import javafx.geometry.Pos ;
6 import javafx.scene.Scene ;
7 import javafx.scene.control.Button ;
8 import javafx.scene.layout.VBox ;
9 import javafx.stage.Stage ;
10 public class TratamentoEventoBotao02 extends Application {
11
12     public static void main(String [] args) {
13         TratamentoEventoBotao02.launch(args) ;
14     }
15
16     @Override
17     public void start(Stage palco) throws Exception {
```

Tratamento de Eventos com Classe Anônima (2)

```
18 VBox painelPrincipal = new VBox(10);
19 painelPrincipal.setAlignment(Pos.CENTER);
20 Button botao1 = new Button("Botao com Tratador em Class Anonima");
21
22 //cria uma instancia a partir de uma classe anonima
23 botao1.setOnAction(
24     new EventHandler() {
25         @Override
26         public void handle(Event event) {
27             System.out.println("\nEvento tratado na classe anonima "
28                 + "\nTipo Evento " + event.getEventType()
29                 + "\nOrigem " + event.getSource());
30         }
31     }
32 );
33
34 painelPrincipal.getChildren().add(botao1);
35 Scene cena = new Scene(painelPrincipal, 300, 200);
36 palco.setScene(cena);
37 palco.setTitle("Tratamento de evento no JavaFX");
38 palco.show();
39 }
40 }
```

Tratamento de Eventos com Classe Anônima (3)



Evento tratado na classe anonima

Tipo Evento ACTION

Origem Button@ba63bbb[styleClass=button]'Botao com Tratador em Class Anonima'

Tratamento de Eventos na Própria Classe (1)

A classe *TratamentoEventoBotao03* define a interface gráfica.

O comportamento do evento é implementado em um método da própria classe *TratamentoEventoBotao03*, que realiza a Interface *EventHandler*.

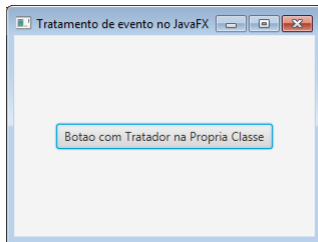
A Interface *EventHandler* permite que o próprio objeto da classe *TratamentoEventoBotao03* possa ser associado ao componente da interface que dispara o evento.

```
1 package visao ;
2 import javafx . application . Application ;
3 import javafx . event . Event ;
4 import javafx . event . EventHandler ;
5 import javafx . geometry . Pos ;
6 import javafx . scene . Scene ;
7 import javafx . scene . control . Button ;
8 import javafx . scene . layout . VBox ;
9 import javafx . stage . Stage ;
10 public class TratamentoEventoBotao03 extends Application implements EventHandler {
11
12     public static void main (String [] args) {
13         TratamentoEventoBotao03 . launch (args) ;
14     }
15 }
```

Tratamento de Eventos na Própria Classe (2)

```
16 @Override
17 public void start(Stage palco) throws Exception {
18     VBox painelPrincipal = new VBox(10);
19     painelPrincipal.setAlignment(Pos.CENTER);
20     Button botao1 = new Button("Botao com Tratador na Propria Classe");
21
22     // cria uma instancia a partir da propria classe da interface grafica
23     botao1.setOnAction(this);
24
25     painelPrincipal.getChildren().add(botao1);
26     Scene cena = new Scene(painelPrincipal, 300, 200);
27     palco.setScene(cena);
28     palco.setTitle("Tratamento de evento no JavaFX");
29     palco.show();
30 }
31
32 @Override
33 public void handle(Event event) {
34     System.out.println("\nEvento tratado no metodo da propria classe "
35         + "\nTipo Evento " + event.getEventType()
36         + "\nOrigem " + event.getSource());
37 }
38 }
```

Tratamento de Eventos na Própria Classe (3)



Evento tratado no metodo da propria classe

Tipo Evento ACTION

Origem Button@5f6eb3eb[styleClass=button]'Botao com Tratador na Propria Classe'

Usando CSS no JavaFX

O uso de CSS no javaFX permite adicionar efeitos, mudar cores, dimensionar e trocar a aparência da aplicação.

CSS é uma linguagem declarativa baseada na identificação de elementos da aplicação e os valores para as propriedades suportadas para os elementos.

Existe duas formas de referenciar um componente no CSS:

- 1 definir no código Java qual a classe do componente e no CSS referenciar a classe com ponto "."
- 2 definir um ID para o componente e referenciar ele usando o caracter "#"

Obs: O ID deve ser único, já a classe pode ser aplicada a diversos componentes.

Usando CSS no JavaFX

Existe duas maneiras de carregar um CSS em uma aplicação JavaFX:

- 1 o CSS pode ficar em um arquivo separado e ser carregado na cena (Scene) da aplicação
- 2 o estilo pode ser adicionado a qualquer classe que estenda Node, por meio do método `setStyle()`

Usando CSS no JavaFX (1)

Exemplo de uso de CSS com inserção no controle por meio do método `setStyle`.

```
1 package visao;  
2  
3 import javafx.application.Application;  
4 import javafx.geometry.HPos;  
5 import javafx.geometry.Insets;  
6 import javafx.scene.Scene;  
7 import javafx.scene.control.Button;  
8 import javafx.scene.control.Label;  
9 import javafx.scene.control.TextField;  
10 import javafx.scene.layout.AnchorPane;  
11 import javafx.scene.layout.BorderPane;  
12 import javafx.scene.layout.ColumnConstraints;  
13 import javafx.scene.layout.GridPane;  
14 import javafx.scene.layout.Priority;  
15 import javafx.scene.paint.Color;  
16 import javafx.scene.text.Font;  
17 import javafx.stage.Stage;  
18  
19 public class ExemploCSS01 extends Application {  
20  
21     public static void main(String[] args) {  
22         ExemploCSS01.launch(args);  
23     }  
24  
25     @Override  
26     public void start(Stage palco) throws Exception {
```

Usando CSS no JavaFX (2)

```
27
28 palco.setTitle("Formulario com Uso de CSS");
29 BorderPane painelPrincipal = new BorderPane();
30 Scene cena = new Scene(painelPrincipal, 380, 170, Color.WHITE);
31
32 String cssLabel = "-fx-font-style: italic;\n" +
33                  "-fx-font-size: 20px;\n" +
34                  "-fx-text-fill: darkgreen;";
35
36 GridPane gridPane = new GridPane();
37 gridPane.setPadding(new Insets(5));
38 gridPane.setHgap(5);
39 gridPane.setVgap(5);
40 ColumnConstraints column1 = new ColumnConstraints(150); //tamanho fixo
41 ColumnConstraints column2 = new ColumnConstraints(50, 130, 400); // min, preferido, max
42 column2.setHgrow(Priority.ALWAYS); //permite redimensionamento
43 gridPane.getColumnConstraints().addAll(column1, column2);
44
45 Label pNomeLabel = new Label("Primeiro Nome");
46 pNomeLabel.setStyle(cssLabel); //usa um bloco de estilo de uma variavel
47
48 TextField pNomeCampo = new TextField();
49
50 Label uNomeLabel = new Label("Ultimo Nome");
51 uNomeLabel.setStyle(cssLabel); //usa um bloco de estilo de uma variavel
52
53 TextField uNomeCampo = new TextField();
54 Button salvarBotao = new Button("Salvar");
55 GridPane.setHalignment(pNomeLabel, HPos.RIGHT);
56 gridPane.add(pNomeLabel, 0, 0);
57 GridPane.setHalignment(uNomeLabel, HPos.RIGHT);
```

Usando CSS no JavaFX (3)

```
58 gridPane.add(uNomeLabel, 0, 1);
59 gridPane.add(pNomeCampo, 1, 0);
60 gridPane.add(uNomeCampo, 1, 1);
61 GridPane.setAlignment(salvarBotao, HPos.RIGHT);
62 gridPane.add(salvarBotao, 1, 2);
63
64 AnchorPane cabecalhoTopo = new AnchorPane();
65 cabecalhoTopo.setPrefHeight(50);
66
67 //define um bloco de estilo diretamente na propriedade style
68 cabecalhoTopo.setStyle("-fx-background-color: linear-gradient("+
69     "from 0% 0% to 100% 100%, blue 0%, silver 100%);");
70
71 Label contatoTexto = new Label("Contatos");
72
73 AnchorPane.setLeftAnchor(contatoTexto, 0.0);
74 AnchorPane.setRightAnchor(contatoTexto, 0.0);
75 contatoTexto.setFont(Font.font("Dialog", 30));
76 contatoTexto.setTextFill(Color.DARKGREEN);
77 cabecalhoTopo.getChildren().add(contatoTexto);
78 painelPrincipal.setTop(cabecalhoTopo);
79 painelPrincipal.setCenter(gridPane);
80
81 palco.setScene(cena);
82 palco.show();
83 }
84 }
```

Usando CSS no JavaFX (1)

Exemplo de uso de CSS com inserção por meio de um arquivo e definição de classes de estilo.

```
1 package visao ;
2 import javafx.application.Application ;
3 import javafx.geometry.HPos ;
4 import javafx.geometry.Insets ;
5 import javafx.scene.Scene ;
6 import javafx.scene.control.Button ;
7 import javafx.scene.control.Label ;
8 import javafx.scene.control.TextField ;
9 import javafx.scene.layout.AnchorPane ;
10 import javafx.scene.layout.BorderPane ;
11 import javafx.scene.layout.ColumnConstraints ;
12 import javafx.scene.layout.GridPane ;
13 import javafx.scene.layout.Priority ;
14 import javafx.scene.paint.Color ;
15 import javafx.stage.Stage ;
16 public class ExemploCSS02 extends Application {
17
18     public static void main(String [] args) {
19         ExemploCSS02.launch(args) ;
20     }
21
22     @Override
23     public void start(Stage palco) throws Exception {
24
25         palco.setTitle("Formulario com Uso de CSS");
26         BorderPane panelPrincipal = new BorderPane();
```

Usando CSS no JavaFX (2)

```
27 Scene cena = new Scene(panelPrincipal, 380, 170, Color.WHITE);
28
29 GridPane gridPane = new GridPane();
30 gridPane.setPadding(new Insets(5));
31 gridPane.setHgap(5);
32 gridPane.setVgap(5);
33 ColumnConstraints column1 = new ColumnConstraints(150); //tamanho fixo
34 ColumnConstraints column2 = new ColumnConstraints(50, 150, 300); // min, preferido, max
35 column2.setHgrow(Priority.ALWAYS); //permite redimensionamento
36 gridPane.getColumnConstraints().addAll(column1, column2);
37
38 Label pNomeLabel = new Label("Primeiro Nome");
39 TextField pNomeCampo = new TextField();
40 Label uNomeLabel = new Label("Ultimo Nome");
41 TextField uNomeCampo = new TextField();
42 Button salvarBotao = new Button("Salvar");
43 Label contatoTexto = new Label("Contatos");
44
45 GridPane.setHalignment(pNomeLabel, HPos.RIGHT);
46 gridPane.add(pNomeLabel, 0, 0);
47 GridPane.setHalignment(uNomeLabel, HPos.RIGHT);
48 gridPane.add(uNomeLabel, 0, 1);
49 GridPane.setHalignment(pNomeCampo, HPos.LEFT);
50 gridPane.add(pNomeCampo, 1, 0);
51 GridPane.setHalignment(uNomeCampo, HPos.LEFT);
52 gridPane.add(uNomeCampo, 1, 1);
53 GridPane.setHalignment(salvarBotao, HPos.RIGHT);
54 gridPane.add(salvarBotao, 1, 2);
55
56 AnchorPane cabecalhoTopo = new AnchorPane();
57 cabecalhoTopo.setPrefHeight(50);
```

Usando CSS no JavaFX (3)

```
58 AnchorPane.setLeftAnchor(contatoTexto, 0.0);
59 AnchorPane.setRightAnchor(contatoTexto, 0.0);
60 cabecalhoTopo.getChildren().add(contatoTexto);
61
62 cabecalhoTopo.getStyleClass().add("painel"); //define uma classe para o CSS
63 contatoTexto.getStyleClass().add("titulo"); //define uma classe para o CSS
64 panelPrincipal.setTop(cabecalhoTopo);
65 panelPrincipal.setCenter(gridPane);
66 cena.getStylesheets().add("/resources/css/emplo01.css");
67 palco.setScene(cena);
68 palco.show();
69 }
70 }
```

Usando CSS no JavaFX (1)

Arquivo CSS com as definições de formatações por classes de estilo.

```
1 .root{
2     -fx-base: darkblue;
3     -fx-background: lightblue;
4 }
5 .button {
6     -fx-font-weight: bold;
7     -fx-font-size: 15px;
8 }
9
10 .label {
11     -fx-font-style: italic;
12     -fx-font-size: 20px;
13     -fx-text-fill: darkgreen;
14 }
15
16 .painel{
17     -fx-background-color: linear-gradient(from 0% 0% to 100% 100%, blue 0%, silver 100%);
18 }
19
20 .titulo {
21     -fx-font-style: normal;
22     -fx-font-weight: bolder;
23     -fx-font-size: 30px;
24     -fx-alignment: CENTER_RIGHT;
25 }
```


JavaFX com FXML

Existem duas maneiras de produção de interface gráfica de usuário com JavaFX.

Usando um arquivo XML ou programando tudo em Java.

As demonstrações anteriores usaram a abordagem de programação em código Java.

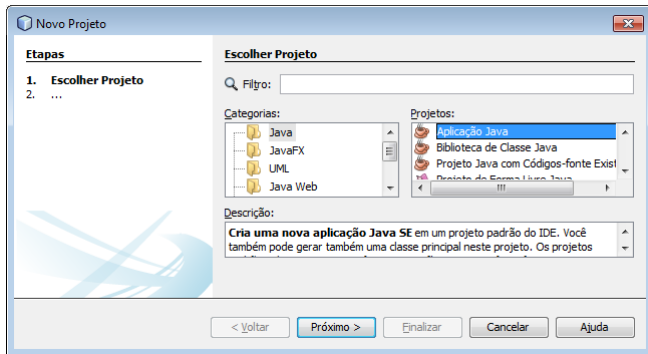
Nesta seção iremos explorar o XML (FXML).

A abordagem do uso do XML para a definição da interface possui duas características importantes que podem ser entendidas como vantagens:

- ❶ A separação do controlador e da visão ficam mais evidente e clara;
- ❷ Pode ser usado o editor gráfico *Scene Builder* para editar o código XML e produzir a interface gráfica.
- ❸ A edição da interface gráfica é realizada separadamente do comportamento da aplicação.

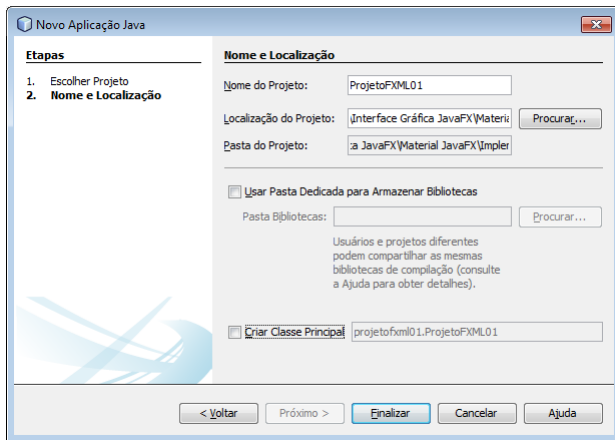
JavaFX com FXML

Criar um Projeto Java do tipo Aplicação Java, conforme dados na tela.



JavaFX com FXML

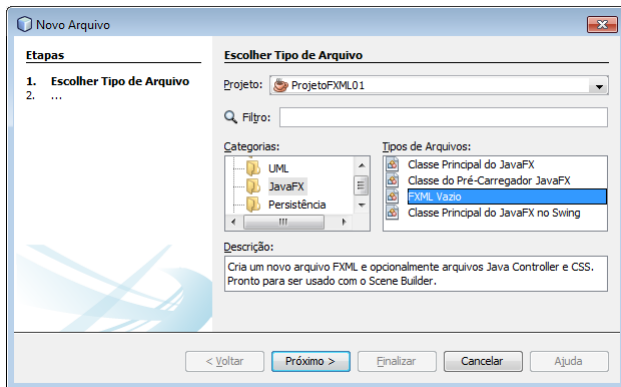
Configure as propriedades conforme dados na tela.



Crie um pacote com o nome **visao**

JavaFX com FXML

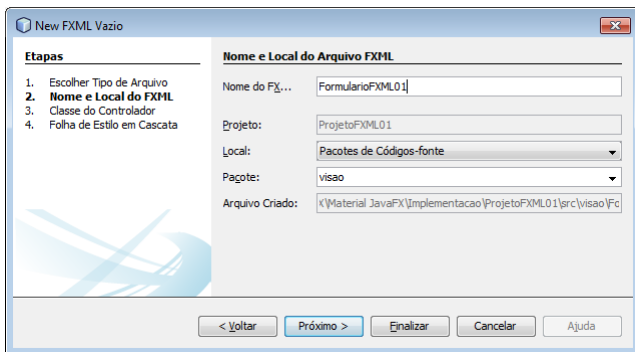
Criar um arquivo FXML Vazio no pacote **visao**, conforme ilustração.



Clique em Próximo...

JavaFX com FXML

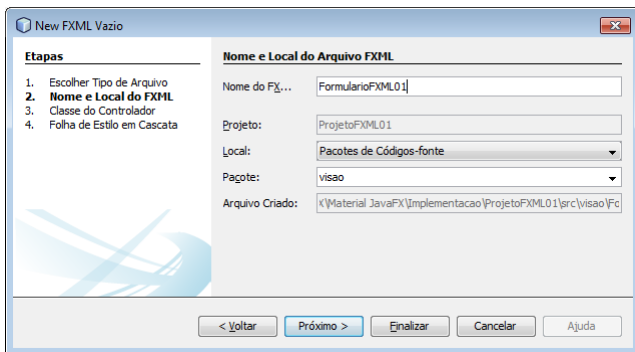
Configure o nome do arquivo FXML e o pacote de destino, conforme ilustração.



Clique em Próximo...

JavaFX com FXML

Configure o nome do arquivo FXML e o pacote de destino, conforme ilustração.

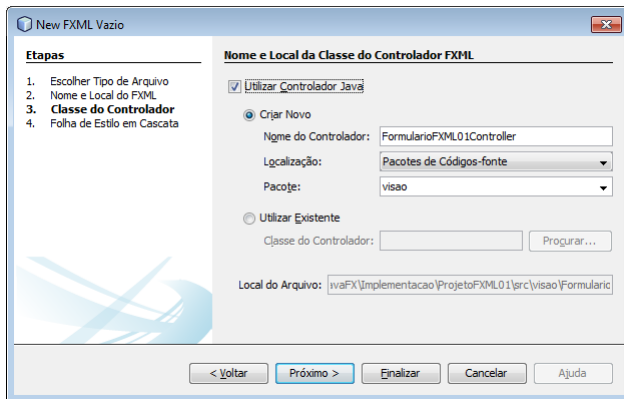


Clique em Próximo...

JavaFX com FXML

Selecione a opção **Utilizar Controlador Java**, selecione a opção **Criar Novo**.

Configure as demais informações conforme ilustração.

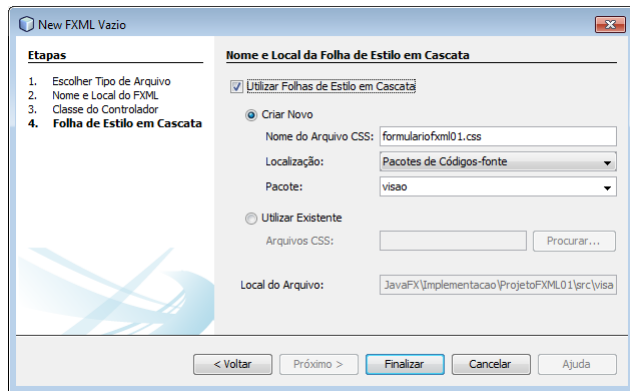


Clique em Próximo...

JavaFX com FXML

Selecione a opção **Utilizar Folhas de Estilo em Cascata**, selecione a opção **Criar Novo**.

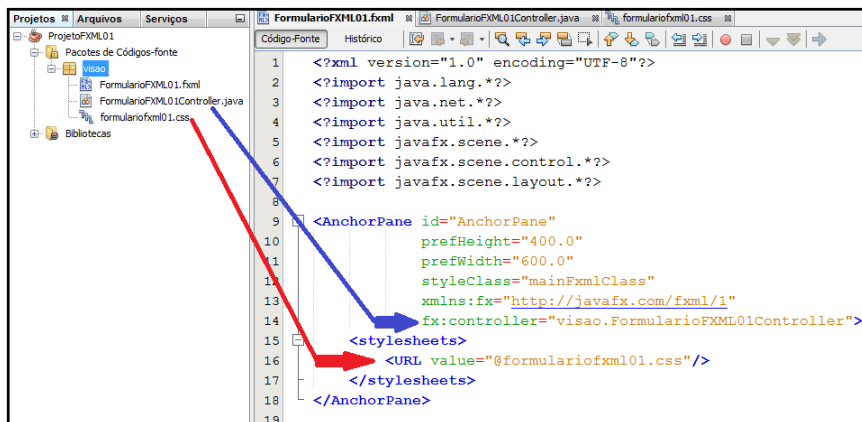
Configure as demais informações conforme ilustração.



Clique em Finalizar...

JavaFX com FXML

Serão criados três arquivos: FormularioFXML01.fxml,
FormularioFXML01Controller.java, formulariofxml01.css



JavaFX com FXML

Arquivo CSS gerado.

```
1  
2 .mainFxmlClass {  
3  
4 }
```

Código 1: Arquivo CSS

JavaFX com FXML (1)

```
1 package visao ;
2 import java.net.URL;
3 import java.util.ResourceBundle;
4 import javafx.event.ActionEvent;
5 import javafx.fxml.FXML;
6 import javafx.fxml.Initializable;
7 import javafx.scene.control.Label;
8 import javafx.scene.control.TextField;
9
10 public class FormularioFXML01Controller implements Initializable {
11     @FXML
12     private TextField campoValor1;
13     @FXML
14     private TextField campoValor2;
15     @FXML
16     private Label campoResultado;
17     @FXML
18     private void botaoSomarOnAction(ActionEvent event) {
19         double valor1, valor2;
20         valor1 = Double.parseDouble(this.campoValor1.getText());
21         valor2 = Double.parseDouble(this.campoValor2.getText());
22         this.campoResultado.setText(String.valueOf(valor1 + valor2));
23     }
24     @FXML
25     private void botaoSairOnAction(ActionEvent event) {
26         System.exit(0);
27     }
28     @Override
29     public void initialize(URL url, ResourceBundle rb) {
30     }
```

JavaFX com FXML (2)

31 | }

Código 2: Controller

JavaFX com FXML

Conteúdo dos arquivos:

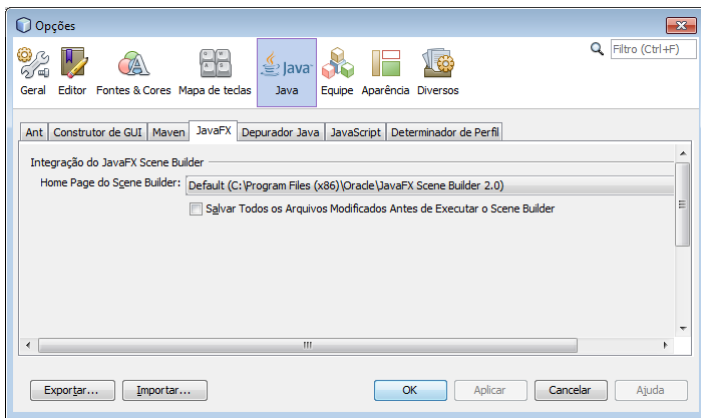
- `FormularioFXML01.fxml` - É o XML que define os componentes e propriedades da interface gráfica;
- `FormularioFXML01Controller.java` - Código Java que implementa as ações da interface (comportamento);
- `formulariofxml01.css` - Folha de estilo para formatar as propriedades visuais da interface;

Nessa modalidade existe uma separação muito forte dos elementos de uma aplicação.

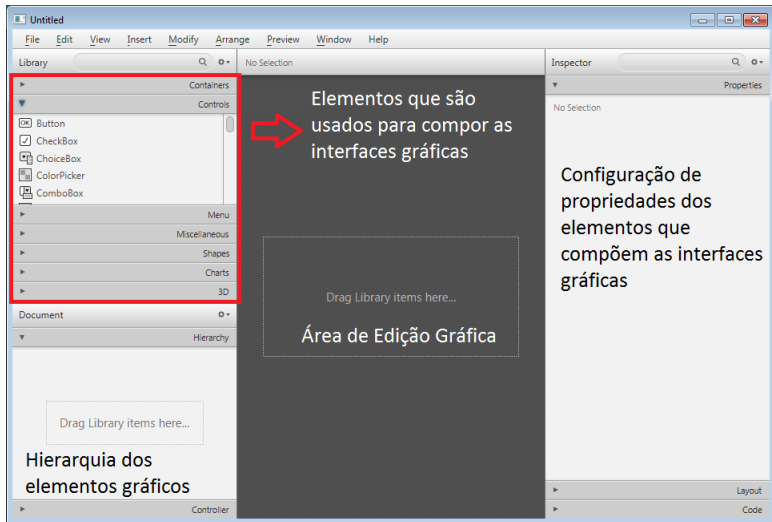
- Componentes e Layout no arquivo FXML;
- Comportamento da aplicação no arquivo Java;
- Aparência dos elementos da interface gráfica no arquivo CSS;
- Pode ser usado o editor gráfico Scene Builder para editar o XML.

Usando Editor Gráfico SceneBuilder

Configurar a integração entre a IDE NetBeans e o editor gráfico Scene Builder



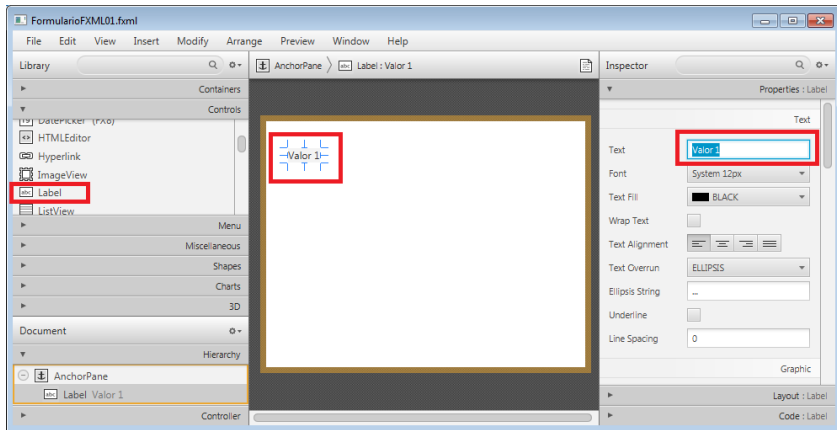
Usando Editor Gráfico SceneBuilder



Usando Editor Gráfico SceneBuilder

Abra o arquivo FormularioFXML01.fxml (na IDE com integração, clique com o botão direito do mouse e escolha a opção Abrir).

Arraste o controle **Label** para a área de desenho e configure a propriedade **Text** conforme ilustração.



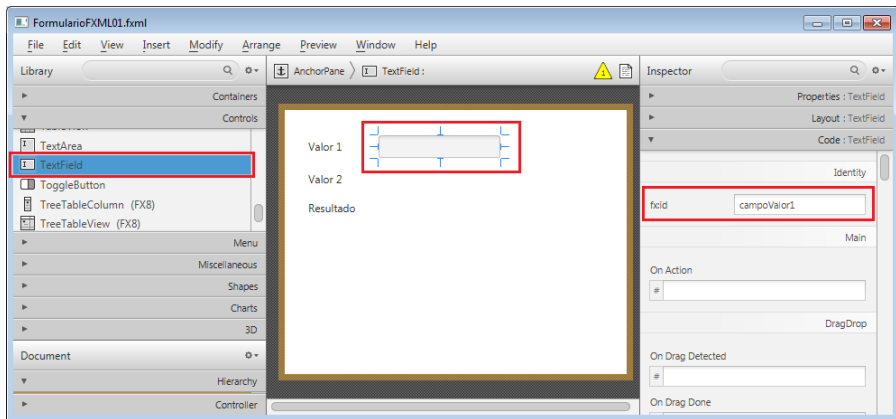
Usando Editor Gráfico SceneBuilder

Repita essa operação para mais dois Label, configurando suas propriedades **Text** conforme ilustração.



Usando Editor Gráfico SceneBuilder

Arraste o controle **TextField** para a área de desenho e configure a propriedade **fx:id** na guia Code, conforme ilustração.

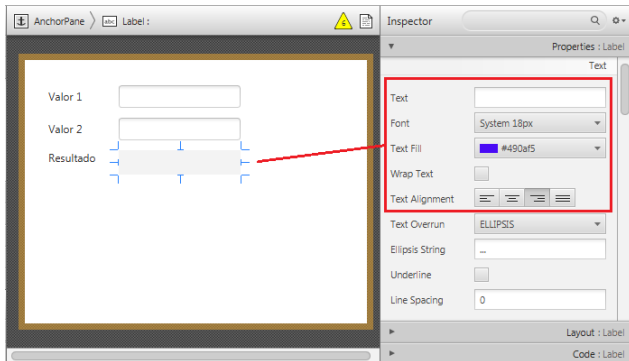


A propriedade **fx:id** é usada no Controller para manipular o componente **TextField** da interface.

Usando Editor Gráfico SceneBuilder

Arraste mais um controle **TextField** para a área de desenho e configure a propriedade **fx:id** com o valor **campoValor2**.

Arraste um novo controle **Label** e configure a propriedade **fx:id** com o valor **campoResultado**. Configure as propriedades **Text**, **Font**, **TextFill**, **Text Alignment** na guia Properties, conforme ilustração.



Usando Editor Gráfico SceneBuilder

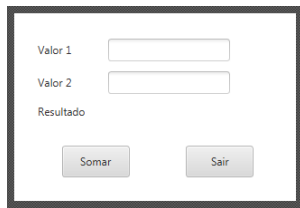
Arraste dois controles **Button** para a área de desenho e configure as propriedades com os seguintes valores:

Botão da Esquerda

- Text : Somar (guia Properties)
- fx:id: botaoSomar (guia Code)

Botão da Direita

- Text : Sair (guia Properties)
- fx:id: botaoSair (guia Code)

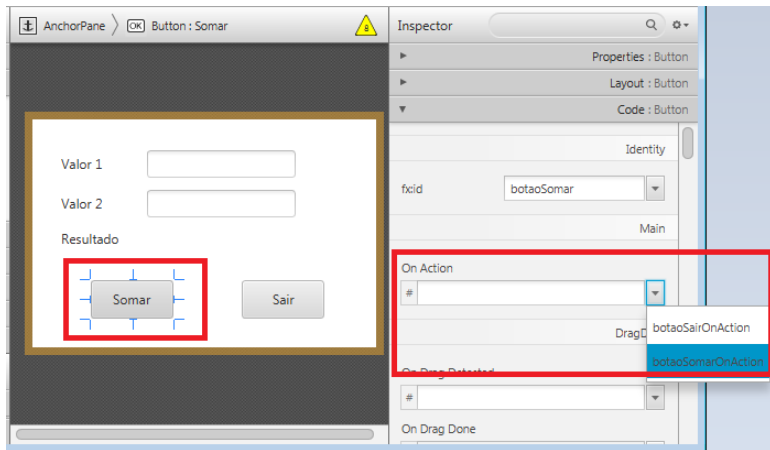


Codificando o Controller

```
1 package visao;
2 import java.net.URL;
3 import java.util.ResourceBundle;
4 import javafx.event.ActionEvent;
5 import javafx.fxml.FXML;
6 import javafx.fxml.Initializable;
7 import javafx.scene.control.Label;
8 import javafx.scene.control.TextField;
9
10 public class FormularioFXML01Controller implements Initializable {
11     @FXML
12     private TextField campoValor1;
13     @FXML
14     private TextField campoValor2;
15     @FXML
16     private Label campoResultado;
17     @FXML
18     private void botaoSomarOnAction(ActionEvent event) {
19         double valor1, valor2;
20         valor1 = Double.parseDouble(this.campoValor1.getText());
21         valor2 = Double.parseDouble(this.campoValor2.getText());
22         this.campoResultado.setText(String.valueOf(valor1 + valor2));
23     }
24     @FXML
25     private void botaoSairOnAction(ActionEvent event) {
26         System.exit(0);
27     }
28     @Override
29     public void initialize(URL url, ResourceBundle rb) {
30     }
31 }
```

Usando Editor Gráfico SceneBuilder

Associar o código do Controller com o componente gráfico no Scene Builder.



Repita esse procedimento para o botão Sair

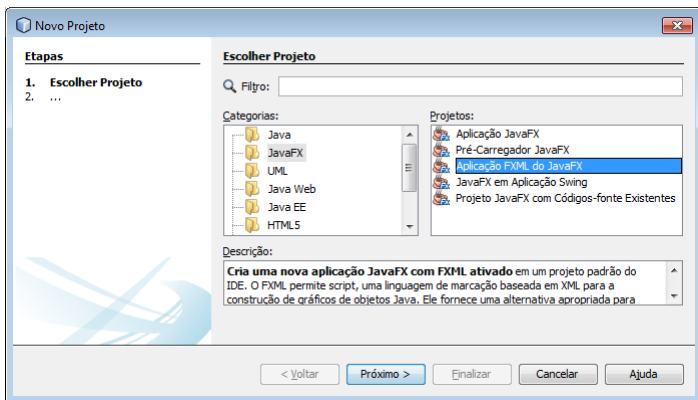
Executando a Aplicação

Carrega o FXML, monta a cena e o palco.

```
1 package visao ;
2 import javafx.application.Application ;
3 import javafx.fxml.FXMLLoader ;
4 import javafx.scene.Parent ;
5 import javafx.scene.Scene ;
6 import javafx.stage.Stage ;
7 public class Principal extends Application {
8
9     @Override
10    public void start(Stage palco) throws Exception {
11        Parent root = FXMLLoader.load(getClass().getResource("FormularioFXML01.fxml"));
12        Scene cena = new Scene(root);
13        palco.setScene(cena);
14        palco.setResizable(false);
15        palco.setTitle("Somar Valores");
16        palco.show();
17    }
18
19    public static void main(String[] args) {
20        launch(args);
21    }
22 }
```

Projeto JavaFXML02

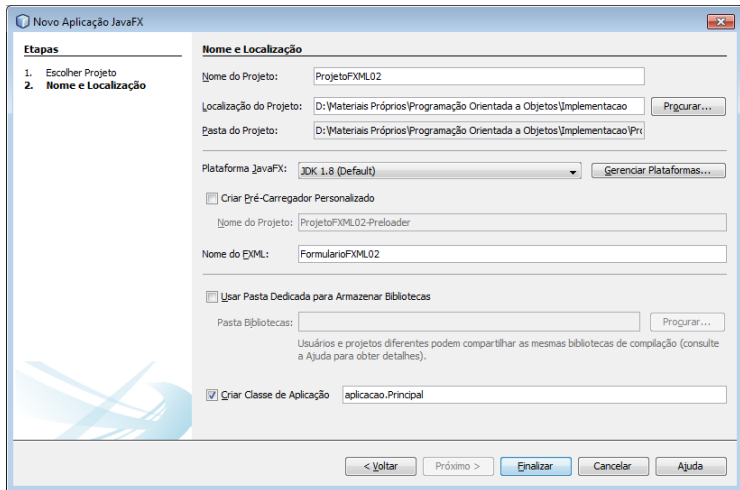
Criar o projeto JavaFX com apoio da IDE. Opção Novo Projeto / JavaFX / Aplicação FXML do JavaFX.



Pressione Próximo

ProjetoJavaFXML02

Preencha as informações do projeto conforme ilustração.



Novo Aplicação JavaFX

Etapas

1. Escolher Projeto
2. **Nome e Localização**

Nome e Localização

Nome do Projeto:

Localização do Projeto:

Pasta do Projeto:

Plataforma JavaFX:

☐ Criar Pré-Carregador Personalizado

Nome do Projeto:

Nome do FXML:

☐ Usar Pasta Dedicada para Armazenar Bibliotecas

Pasta Bibliotecas:

Usuários e projetos diferentes podem compartilhar as mesmas bibliotecas de compilação (consulte a Ajuda para obter detalhes).

☒ Criar Classe de Aplicação

Pressione Finalizar

ProjetoJavaFXML02

A IDE gera três arquivos: FXML, Controller e a Application

```
<?xml version="1.0" encoding="UTF-8"?>
<?import java.lang.*?>
<?import java.util.*?>
<?import javafx.scene.*?>
<?import javafx.scene.control.*?>
<?import javafx.scene.layout.*?>
<AnchorPane id="AnchorPane"
    prefHeight="200"
    prefWidth="320"
    xmlns:fx="http://javafx.com/fxml/1"
    fx:controller="aplicacao.FormularioFXML02Controller">
    <children>
        <Button layoutX="126" layoutY="90" text="Click Me!"
            onAction="#handleButtonAction" fx:id="button" />
        <Label layoutX="126" layoutY="120" minHeight="16"
            minWidth="69" fx:id="label" />
    </children>
</AnchorPane>
```

ProjetoJavaFXML02

```
package aplicacao;
import ...6 linhas

public class FormularioFXML02Controller implements Initializable {

    @FXML
    private Label label;

    @FXML
    private void handleButtonAction(ActionEvent event) {
        System.out.println("You clicked me!");
        label.setText("Hello World!");
    }

    @Override
    public void initialize(URL url, ResourceBundle rb) {
        // TODO
    }
}
```

ProjetoJavaFXML02

```
package aplicacao;
import ...5 linhas
public class Principal extends Application {

    @Override
    public void start(Stage stage) throws Exception {
        Parent root = FXMLLoader.load(
            getClass().getResource("FormularioFXML02.fxml"));
        Scene scene = new Scene(root);
        stage.setScene(scene);
        stage.show();
    }

    public static void main(String[] args) {
        launch(args);
    }
}
```

ProjetoJavaFXML02 (1)

Editar o código da classe ProjetoFXML02Controller para o código a seguir.

```
1 package aplicacao;
2 import java.net.URL;
3 import java.util.ResourceBundle;
4 import javafx.event.ActionEvent;
5 import javafx.fxml.FXML;
6 import javafx.fxml.Initializable;
7 import javafx.scene.control.Label;
8 import javafx.scene.control.TextField;
9 public class FormularioFXML02Controller implements Initializable {
10
11     @FXML
12     private TextField campoValor1;
13     @FXML
14     private TextField campoValor2;
15     @FXML
16     private Label campoResultado;
17
18     @FXML
19     private void botaoSomarOnAction(ActionEvent event) {
20         double valor1, valor2;
21         try {
22             valor1 = Double.parseDouble(this.campoValor1.getText());
23             valor2 = Double.parseDouble(this.campoValor2.getText());
24             this.campoResultado.setText(String.valueOf(valor1 + valor2));
25         } catch (Exception ex) {
26             System.out.println(ex.getMessage());
27         }
28     }
```

ProjetoJavaFXML02 (2)

```
29
30 @FXML
31 private void botaoSairOnAction(ActionEvent event) {
32     System.exit(0);
33 }
34
35 @Override
36 public void initialize(URL url, ResourceBundle rb) {
37 }
38 }
```

ProjetoJavaFXML02

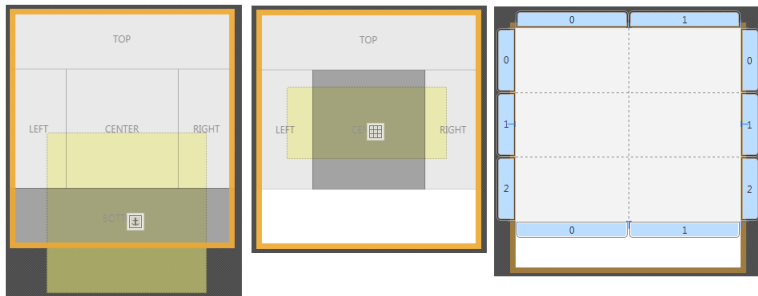
Abrir o arquivo FormularioFXML02.fxml no editor Scene Builder para editar o layout da interface.

Selecione e remova o AnchorPane criado pela IDE.

Arraste um BorderPane para a área de edição.

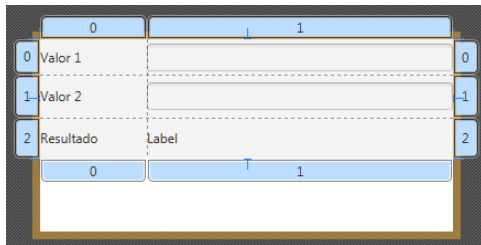
Arraste um AnchorPane para a região Bottom do BorderPane.

Arraste um GridPane para a região Center do BorderPane.



ProjetoJavaFXML02

Inserir os controles Label e TextField conforme ilustração da tela.



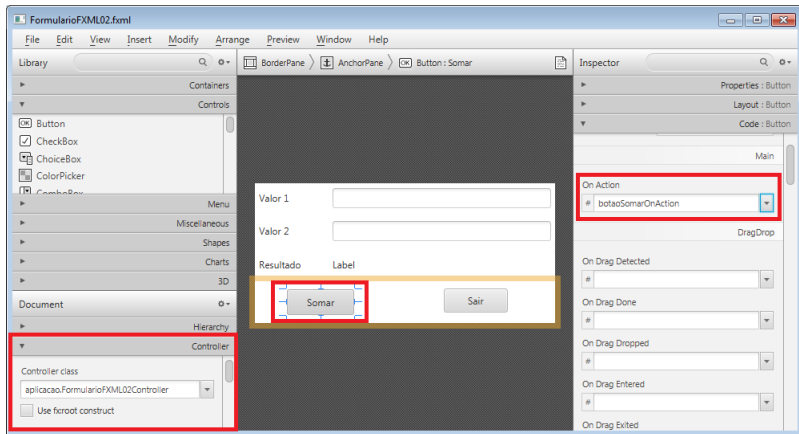
Configure as propriedades dos Labels da zero do GridPane conforme imagem;

Configure a propriedade `fx:id` dos componentes da coluna da direita: `campoValor1`, `campoValor2` e `campoResultado` respectivamente;

ProjetoJavaFXML02

Associar o Controller ao FXML;

Associar os comportamentos escritos no Controller com os botões: botaoSomar e botaoSair.



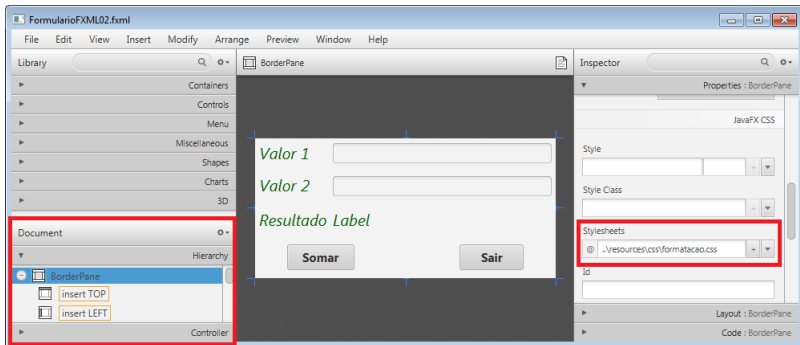
ProjetoJavaFXML02 (1)

Arquivo CSS a ser associado ao formulário em edição no Scene Builder.

```
1 .root{
2     -fx-base: darkblue;
3     -fx-background: lightblue;
4 }
5 .button {
6     -fx-font-weight: bold;
7     -fx-font-size: 15px;
8 }
9
10 .label {
11     -fx-font-style: italic;
12     -fx-font-size: 20px;
13     -fx-text-fill: darkgreen;
14 }
15
16 .resultado {
17     -fx-font-style: normal;
18     -fx-font-weight: bolder;
19     -fx-font-size: 20px;
20     -fx-alignment: CENTER_RIGHT;
21     -fx-background-color: silver;
22 }
```

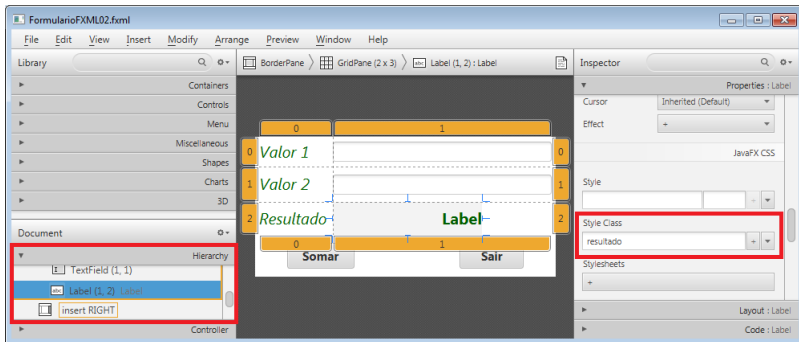
ProjetoJavaFXML02

Associar o CSS ao painel BorderPane (principal)



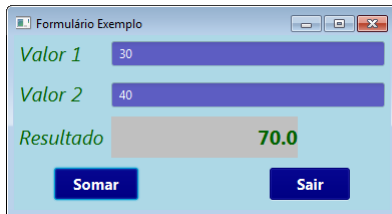
ProjetoJavaFXML02

Definir a classe de formatação para o campoResultado.



ProjetoJavaFXML02

Resultado da Execução.



Formulário Exemplo

Valor 1 30

Valor 2 40

Resultado 70.0

Somar Sair